**Module 3: Types et constructions de base de Visual C #**

# **Contenu:**

|  |  |
| --- | --- |
|  | [Présentation du module](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C03.html#P1) |
| **Leçon 1:** | [Implémentation de structures et d'énumérations](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C03.html#P2) |
| **Leçon 2:** | [Organisation des données en collections](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C03.html#P3) |
| **Lecon 3:** | [Gestion des événements](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C03.html#P4) |
| **Laboratoire:** | [Rédaction du code pour l'application de prototype de grades](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C03.html#P5) |
|  | [Revue du module et points à retenir](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C03.html#P6) |

# ****Table des matieres****

[I. Contenu: 1](#_Toc68761670)

[Table des matieres 2](#_Toc68761671)

[II. Présentation du module 5](#_Toc68761672)

[A. Objectifs 5](#_Toc68761673)

[III. Leçon 1: Implémentation de structures et d'énumérations 5](#_Toc68761674)

[A. Objectifs de la leçon 6](#_Toc68761675)

[B. Créer et utiliser des énumérations 6](#_Toc68761676)

[C. Créer et utiliser des structures 8](#_Toc68761677)

[1. Créer une structure 9](#_Toc68761678)

[a) Déclarer une structure 9](#_Toc68761679)

[2. Utiliser un Struct 10](#_Toc68761680)

[a) Instancier une structure 10](#_Toc68761681)

[D. Initialisation des structures 10](#_Toc68761682)

[E. Créer des propriétés 12](#_Toc68761683)

[F. Création d'indexeurs 18](#_Toc68761684)

[G. Démonstration: création et utilisation d'une structure 21](#_Toc68761685)

[1. Démonstration étapes 21](#_Toc68761686)

[IV. Leçon 2: Organisation des données en collections 21](#_Toc68761687)

[A. Objectifs de la leçon 22](#_Toc68761688)

[B. Choisir les collections 22](#_Toc68761689)

[C. Classes de collecte standard 24](#_Toc68761690)

[D. Classes de collection spécialisées 25](#_Toc68761691)

[E. Utilisation des collections de listes 28](#_Toc68761692)

[F. Utilisation des collections de dictionnaires 30](#_Toc68761693)

[G. Interroger une collection 32](#_Toc68761694)

[V. Leçon 3: Gestion des événements 35](#_Toc68761695)

[A. Objectifs de la leçon 36](#_Toc68761696)

[B. Délégués et événements - Introduction 36](#_Toc68761697)

[C. Création d'événements et de délégués 38](#_Toc68761698)

[D. Élever des événements 40](#_Toc68761699)

[E. S'abonner à des événements 43](#_Toc68761700)

[F. Démonstration: utilisation des événements en XAML 45](#_Toc68761701)

[1. Démonstration étapes 45](#_Toc68761702)

[G. Démonstration: rédaction de code pour le laboratoire d'application du prototype de grades 45](#_Toc68761703)

[1. Démonstration étapes 46](#_Toc68761704)

[VI. Laboratoire: Rédaction du code pour l'application de prototype Grades 46](#_Toc68761705)

[1. Scénario 46](#_Toc68761706)

[2. Objectifs 46](#_Toc68761707)

[3. Exercice 1: Ajout d'une logique de navigation à l'application de prototype de grades 47](#_Toc68761708)

[4. Exercice 2: Création de types de données pour stocker les informations sur les utilisateurs et les notes 48](#_Toc68761709)

[5. Exercice 3: Affichage des informations sur l'utilisateur et la note 48](#_Toc68761710)

[VII. Revue du module et points à retenir 49](#_Toc68761711)

[1. Question (s) de révision 49](#_Toc68761712)

# **Présentation du module**

Pour créer des applications efficaces à l'aide de Windows Presentation Foundation (WPF) ou d'autres plates-formes .NET Framework, vous devez d'abord apprendre quelques constructions Visual C # de base. Vous devez savoir comment créer des structures simples pour représenter les éléments de données avec lesquels vous travaillez. Vous devez savoir comment organiser ces structures en collections, afin de pouvoir ajouter des éléments, récupérer des éléments et parcourir vos éléments. Enfin, vous devez savoir comment vous abonner aux événements afin de pouvoir répondre aux actions de vos utilisateurs.

Dans ce module, vous apprendrez à créer et à utiliser des structures et des énumérations, à organiser les données en collections et à créer et vous abonner à des événements.

## ****Objectifs****

Après avoir terminé ce module, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Créez et utilisez des structures et des énumérations. |
| • | Utilisez des classes de collection pour organiser les données. |
| • | Créez et abonnez-vous à des événements. |

# Leçon 1: Implémentation de structures et d'énumérations

Le .NET Framework comprend divers types de données intégrés, tels que Int32, Decimal, String et Boolean. Cependant, supposons que vous souhaitiez créer un objet représentant un café. Quel type utiliseriez-vous? Vous pouvez utiliser des types intégrés pour représenter les propriétés d'un café, comme le pays d'origine (une chaîne) ou la force du café (un entier). Cependant, vous avez besoin d'un moyen de représenter le café comme une entité discrète, de sorte que vous puissiez effectuer des actions telles que l'ajout d'un café à une collection ou la comparaison d'un café à un autre.

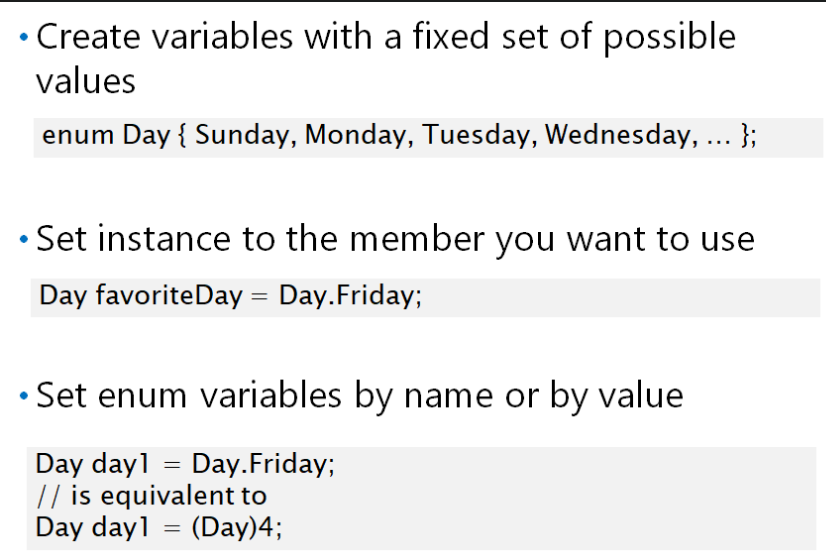
Dans cette leçon, vous apprendrez à utiliser des structures et des énumérations pour créer vos propres types simples.

## ****Objectifs de la leçon****

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Créez et utilisez des énumérations. |
| • | Créez et utilisez des structures. |
| • | Définissez des constructeurs pour instancier des structures. |
| • | Créez des propriétés pour obtenir et définir des valeurs de champ dans une structure. |
| • | Créez des indexeurs pour exposer les membres de la structure à l'aide d'un index entier. |

## ****Créer et utiliser des énumérations****



**Un type d'énumération, ou enum**, est une structure qui vous permet de créer une variable avec un ensemble fixe de valeurs possibles. L'exemple le plus courant consiste à utiliser une énumération pour définir le jour de la semaine. Il n'y a que sept valeurs possibles pour les jours de la semaine et vous pouvez être raisonnablement certain que ces valeurs ne changeront jamais.

L'exemple suivant montre comment créer une énumération:

**Déclarer une énumération**

enum Day {dimanche, lundi, mardi, mercredi, jeudi, vendredi, samedi};

Pour utiliser l'énumération, vous créez une instance de votre variable d'énumération et spécifiez le membre d'énumération que vous souhaitez utiliser.

L'exemple suivant montre comment utiliser une énumération:

**Utiliser une énumération**

Day favoriteDay = Day.Friday;

L'utilisation d'énumérations présente plusieurs avantages par rapport à l'utilisation de types textuels ou numériques:

|  |  |
| --- | --- |
| • | Gérabilité améliorée. En contraignant une variable à un ensemble fixe de valeurs valides, vous êtes moins **susceptible de rencontrer des arguments non valides et des fautes d'orthographe.** |
| • | Amélioration de l'expérience des développeurs. Dans Visual Studio, **la fonctionnalité IntelliSense vous invite avec les valeurs disponibles lorsque vous utilisez une énumération.** |
| • | **Meilleure lisibilité du code. La syntaxe enum facilite la lecture et la compréhension de votre code.** |

**Chaque membre d'une énumération a un nom et une valeur**. Le nom est la chaîne que vous définissez entre accolades, comme dimanche ou lundi. Par défaut, la valeur est un entier. Si vous ne spécifiez pas de valeur pour chaque membre, les membres reçoivent des valeurs incrémentielles **commençant par 0**. Par exemple, Day.Sunday est égal à 0 et Day.Monday est égal à 1.

L'exemple suivant montre comment utiliser les noms et les valeurs de manière interchangeable:

**Utilisation interchangeable de noms et de valeurs d'énumération**

// Définit une variable d'énumération par son nom.

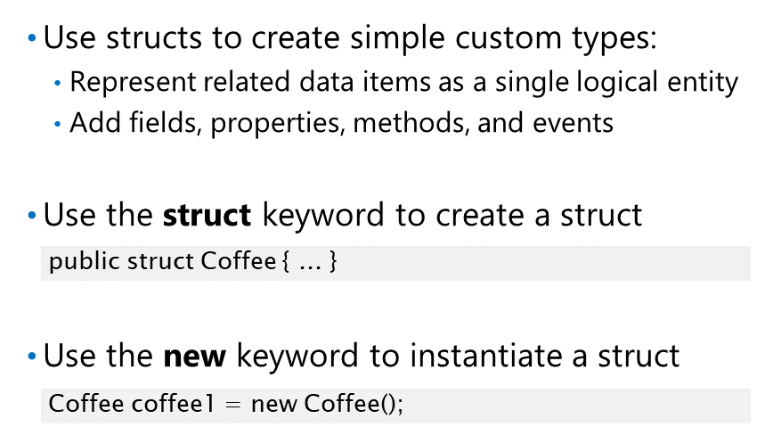
Day favoriteDay = Day.Friday;

// Définit une variable d'énumération par valeur.

Day favoriteDay = (Day) 4;

**Liens de référence:**Pour plus d'informations sur les énumérations, consultez la page Types d'énumération (Guide de programmation C #) à l'adresse<https://aka.ms/moc-20483c-m3-pg1>.

## ****Créer et utiliser des structures****



Dans Visual C #, **une structure** est une construction de programmation que vous **pouvez utiliser pour définir des types personnalisés.** Les structures sont essentiellement des structures de données légères qui représentent des informations connexes sous la forme d'un élément unique. Par exemple:

|  |  |
| --- | --- |
| • | Une structure nommée Point peut être constituée de champs pour représenter une coordonnée x et une coordonnée y. |
| • | Une structure nommée Circle peut être constituée de champs pour représenter une coordonnée x, une coordonnée y et un rayon. |
| • | Une structure nommée Color peut être constituée de champs pour représenter un composant rouge, un composant vert et un composant bleu. |

**La plupart des types intégrés dans Visual C #, tels que int, bool et char, sont définis par structs**. Vous pouvez utiliser des structures pour créer vos propres types qui se comportent comme des types intégrés.

### ****Créer une structure****

Vous utilisez le **mot clé struct** pour déclarer une structure, comme le montre l'exemple suivant: You use the struct keyword to declare a struct, as shown by the following example:

#### *Déclarer une structure*

public struct Coffee

{

public int Strength;

public string Bean;

chaîne publique CountryOfOrigin;

// Autres méthodes, champs, propriétés et événements.

}

Le **mot clé struct** est précédé d'un **modificateur d'accès - public** dans l'exemple ci-dessus - qui spécifie où vous pouvez utiliser le type. Vous pouvez utiliser les modificateurs d'accès suivants dans vos déclarations de structure:

| **Modificateur d'accès** | **Des détails** |
| --- | --- |
| **public** | Le type est disponible pour le code exécuté dans n'importe quel assembly. |
| **internal** | Le type est disponible pour n'importe quel code dans le même assembly, mais pas disponible pour coder dans un autre assembly. Il s'agit de la **valeur par défaut** **si vous ne spécifiez pas de modificateur d'accès.** |
| **private** | Le type est uniquement disponible pour coder dans la structure qui le contient. Vous ne pouvez utiliser le modificateur d'accès privé qu'avec des structures imbriquées. |

Les **structures** peuvent contenir une variété de **membres**, y compris des **champs**, des **propriétés**, des **méthodes** et des **événements**.

### ****Utiliser un Struct****

Pour **créer une instance d'une structure**, vous utilisez le **mot clé new**, comme illustré par l'exemple suivant:

#### *Instancier une structure*

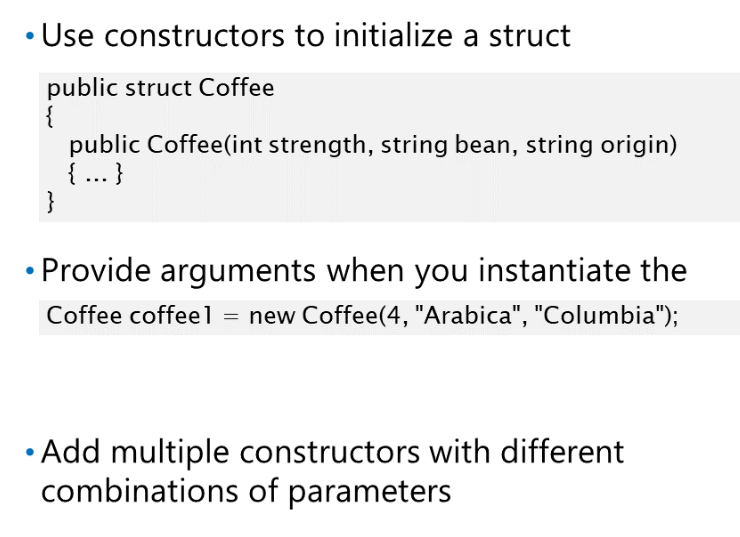
Coffee coffee1 = new Coffee();

coffee1.Strength = 3;

coffee1.Bean = "Arabica";

coffee1.CountryOfOrigin = "Kenya";

## ****Initialisation des structures****

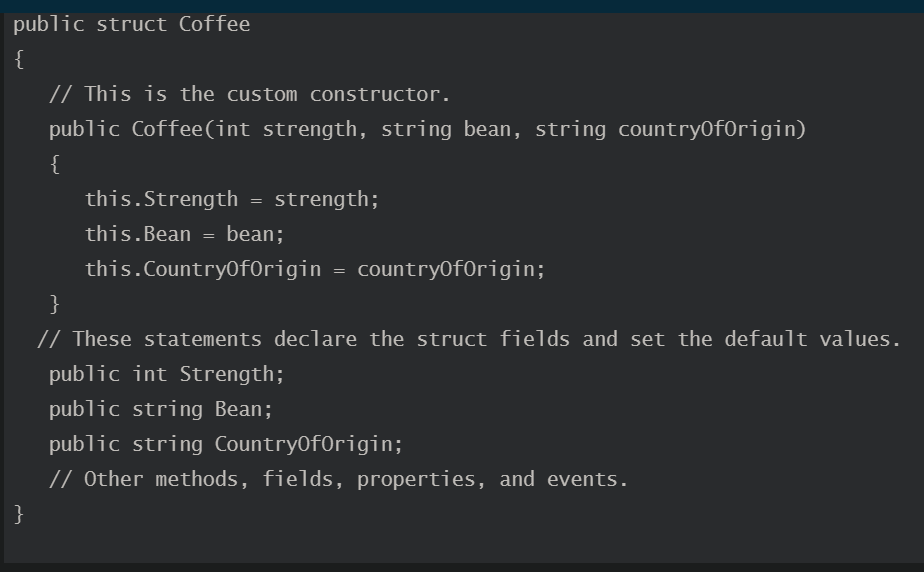


Vous avez peut-être remarqué que la syntaxe pour instancier une structure, par exemple new Coffee (), est similaire à la syntaxe pour appeler une méthode. En effet, lorsque vous instanciez une structure, vous appelez en fait un type spécial de méthode appelé constructeur. **Un constructeur est une méthode de la structure qui porte le même nom que la structure.**

**Lorsque vous instanciez une structure sans arguments, comme new Coffee (), *vous appelez le constructeur par défaut* qui est créé par le compilateur Visual C #.** Si vous souhaitez pouvoir spécifier des valeurs de champ par défaut lorsque vous instanciez une structure, vous pouvez ajouter des constructeurs qui acceptent des paramètres à votre structure.

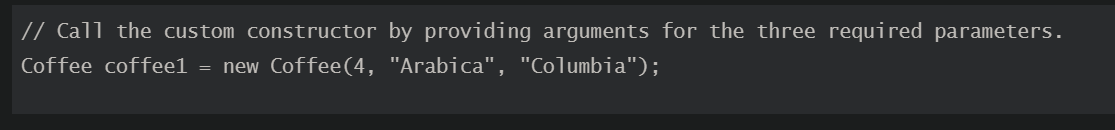
L'exemple suivant montre comment créer un constructeur dans une structure:

**Ajouter un constructeur**



L'exemple suivant montre comment utiliser ce constructeur pour instancier un élément Coffee:

**Appeler un constructeur**



, chaque constructeur acceptant une combinaison différente de paramètres. Cependant, vous ne pouvez pas ajouter un constructeur par défaut à une structure car il est créé par le compilateur.

## ****Créer des propriétés****



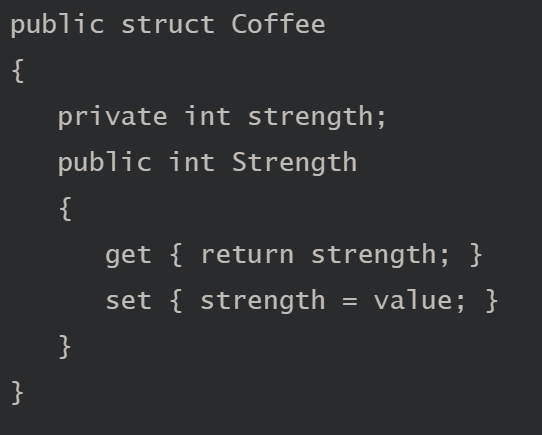
Dans Visual C #, **une propriété** est une construction de programmation qui permet au code client **d'obtenir ou de définir la valeur de champs privés** dans une structure ou une classe.

Pour les consommateurs de votre structure ou classe, **la propriété se comporte comme un champ public**. Dans votre structure ou classe, la **propriété** **est implémentée à l'aide d'accesseurs**, qui sont un **type spécial de méthode**. Une propriété peut inclure l'un des éléments suivants ou les deux:

|  |  |
| --- | --- |
| • | Un **accesseur get** pour fournir un accès en **lecture** à un champ. |
| • | Un **accesseur set** pour fournir un accès en **écriture** à un champ. |

L'exemple suivant montre comment implémenter une propriété dans une structure:

**Implémentation d'une propriété**

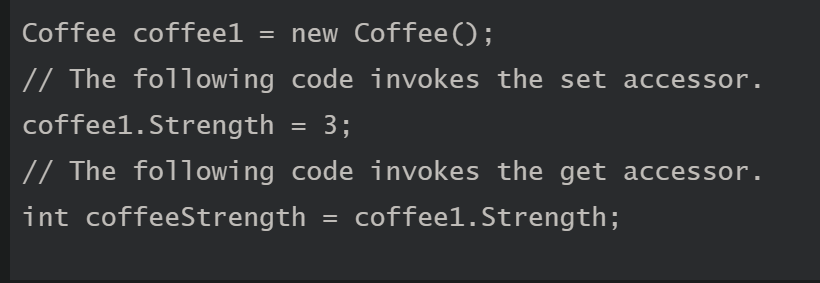


Dans la propriété, les accesseurs get et set utilisent la syntaxe suivante:

|  |  |
| --- | --- |
| • | L'accesseur **get** utilise le mot-clé **return** pour renvoyer la valeur du champ privé à l'appelant. |
| • | L'accesseur **set** utilise une **variable locale spéciale nommée value** pour définir la valeur du champ privé. La variable value contient la valeur fournie par le code client lors de l'accès à la propriété. |

L'exemple suivant montre comment utiliser une propriété:

**Utiliser une propriété**



Le code client utilise la propriété comme s'il s'agissait d'un champ public. Cependant, l'utilisation de propriétés publiques pour exposer des champs privés offre les avantages suivants par rapport à l'utilisation directe des champs publics:

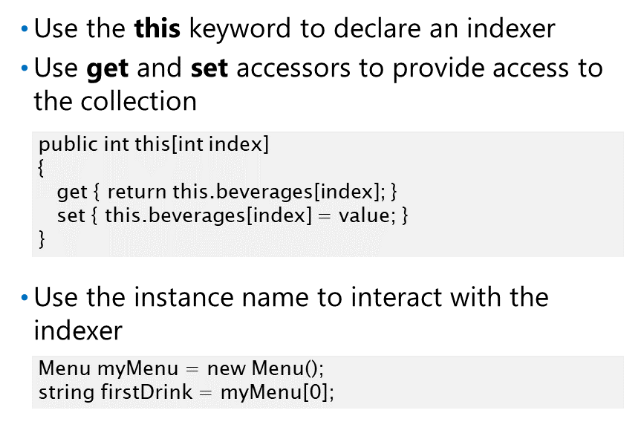
|  |  |  |  |
| --- | --- | --- | --- |
| • | Vous pouvez utiliser des propriétés pour contrôler l'accès externe à vos champs. Une propriété qui inclut uniquement un accesseur get est en lecture seule, tandis qu'une propriété qui inclut uniquement un accesseur set est en écriture seule.  // Ceci est une propriété en lecture seule.   |  |  | | --- | --- | | • | You can use properties to control external access to your fields. A property that includes only a get accessor is read-only, while a property th  at includes only a set accessor is write-only.  // This is a read-only property.  public int Strength  {  get { return strength; }  }  // This is a write-only property.  public string Bean  {  set { bean = value; }  } | |
| • | Vous pouvez modifier l'implémentation des propriétés sans affecter le code client. Par exemple, vous pouvez ajouter une logique de validation ou appeler une méthode au lieu de lire une valeur de champoplementation of properties without affecting client code. For example, you can add validation logic, or call a method instead of reading a field value.  public int Strength{  get { return strength; }  set  {  if(value < 1)  { strength = 1; }  else if(value > 5)  { strength = 5; }  else  { strength = value; }  }  } |
| • | Vous pouvez également créer une propriété const en renvoyant simplement une valeur littérale.  public string Hello { get { return “world”; }} |
| • | Les propriétés sont requises pour la liaison de données dans WPF. Par exemple, vous pouvez lier des contrôles à des valeurs de propriété, mais vous ne pouvez pas lier des contrôles à des valeurs de champ.  Lorsque vous souhaitez créer une propriété qui obtient et définit simplement la valeur d'un champ privé sans effectuer de logique supplémentaire, **vous pouvez utiliser une syntaxe abrégée.** |
| • | Pour créer une propriété qui lit et écrit dans un champ privé, vous pouvez utiliser la syntaxe suivante:  public int Strength { get; set; } |
| • | Pour créer une propriété qui peut être lue publiquement, mais définie uniquement par sa classe contenant, vous pouvez utiliser la syntaxe suivante:  public int Strength { get; private set; } |
| • | Pour créer une propriété en lecture seule dans laquelle seul le constructeur peut définir la valeur sur cette propriété, vous pouvez utiliser la syntaxe suivante:  public int Strength { get; } |
| • | Pour créer une propriété qui écrit dans un champ privé, vous pouvez utiliser  la syntaxe suivante: |

public int Strength { private get; set; }

Dans chaque cas, le **compilateur créera implicitement un champ privé** et le **mappera à votre propriété**. Celles-ci sont appelées **propriétés implémentées automatiquement**. Vous pouvez modifier l'implémentation de votre propriété à tout moment.

**Lecture supplémentaire:**En plus de contrôler l'accès à une propriété en omettant les accesseurs get ou set, vous pouvez également restreindre l'accès en appliquant des modificateurs d'accès (tels que privé ou protégé) à vos accesseurs. Par exemple, vous pouvez créer une propriété avec un accesseur get public et un accesseur set protégé. Pour plus d'informations, consultez la page Restricting Accessor Accessibility (C # Programming Guide) à l'adresse<https://aka.ms/moc-20483c-m3-pg2>.

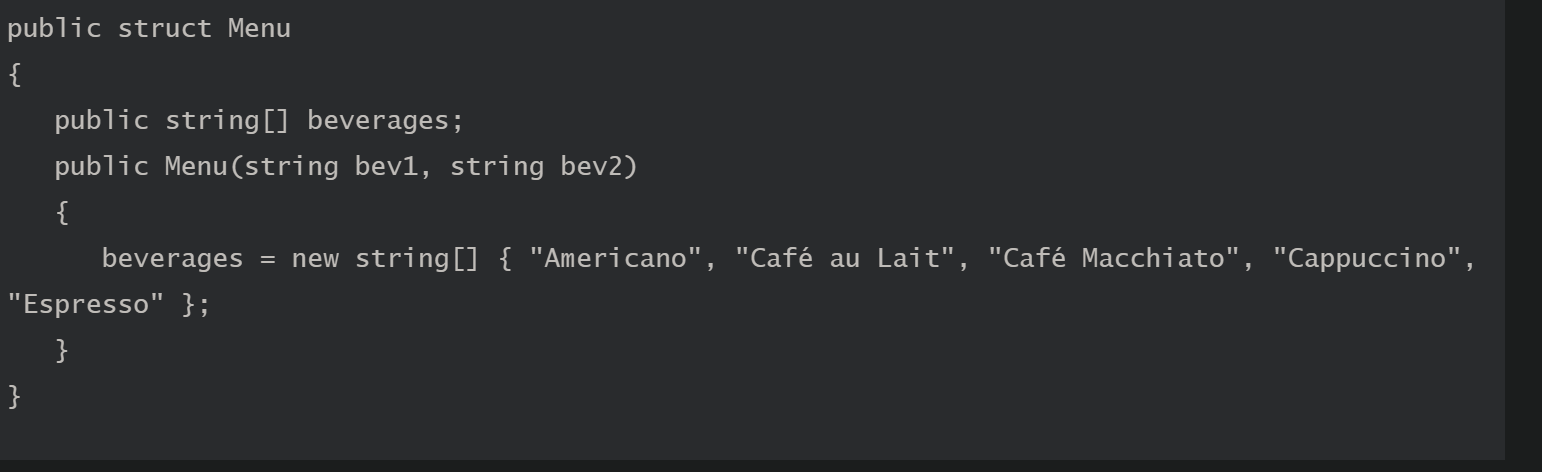
## ****Création d'indexeurs****



Dans certains scénarios, vous souhaiterez peut-être utiliser une structure ou une classe comme conteneur pour un tableau de valeurs. Par exemple, vous pouvez créer une structure pour représenter les boissons disponibles dans un café. La structure peut utiliser un tableau de chaînes pour stocker la liste des boissons.

L'exemple suivant montre une structure qui inclut un tableau:

**Création d'une structure qui inclut un tableau**



Lorsque vous exposez le tableau en tant que champ public, vous utilisez la syntaxe suivante pour récupérer les boissons de la liste:

**Accès direct aux éléments de la baie**

Menu myMenu = new Menu();

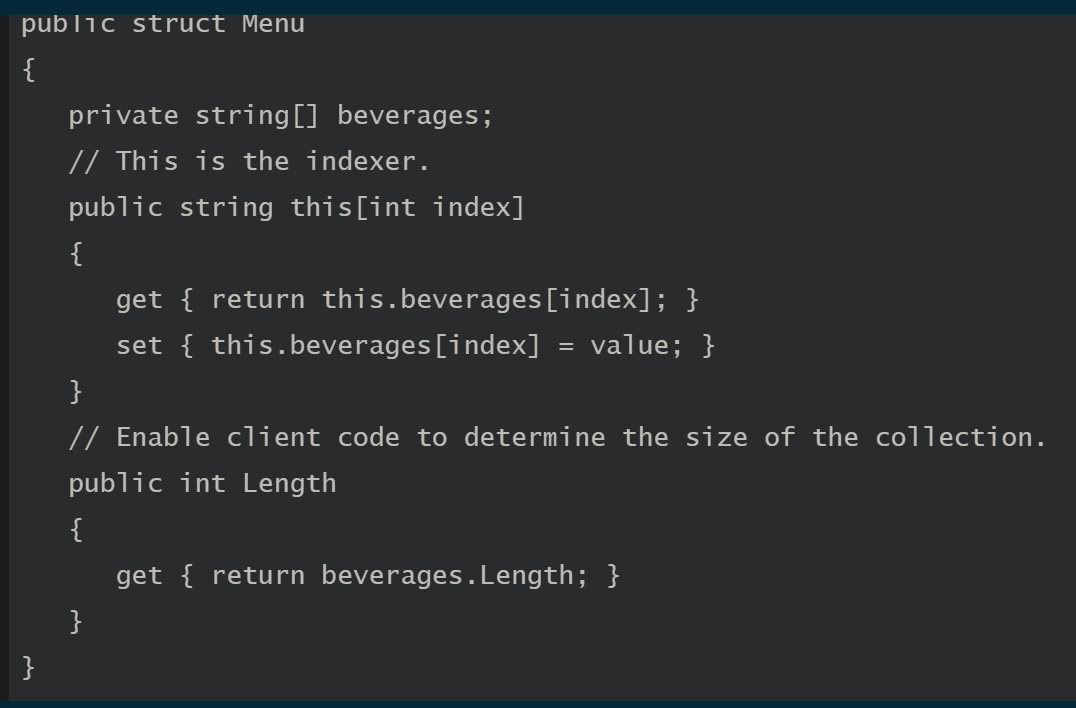
string firstDrink = myMenu.beverages[0];

Une approche plus intuitive serait si vous pouviez accéder au premier élément du menu en utilisant **la syntaxe myMenu [0]**. Vous pouvez le faire en créant **un indexeur**.

**Un indexeur est similaire à une propriété**, en ce sens qu'il utilise des **accesseurs get et set pour contrôler l'accès à un champ**. Plus important encore, **un indexeur vous permet d'accéder aux membres de la collection directement à partir du nom de la structure ou de la classe** contenant en fournissant une valeur d'index entière. Pour **déclarer un indexeur**, vous utilisez le **mot clé this**, qui indique que la **propriété sera accessible en utilisant le nom de l'instance de struct.**

L'exemple suivant montre comment définir un indexeur pour une structure:

**Créer un indexeur**



Lorsque vous utilisez un indexeur pour exposer le tableau, vous utilisez la syntaxe suivante pour récupérer les boissons de la liste:

**Accès aux éléments de tableau à l’aide d’un indexeur**

Menu myMenu = new Menu();

string firstDrink = myMenu[0];

int numberOfChoices = myMenu.Length;

Tout comme une propriété, vous pouvez personnaliser les accesseurs get et set dans un indexeur sans affecter le code client. Vous pouvez créer un indexeur en lecture seule en incluant uniquement un accesseur get, et vous pouvez créer un indexeur en écriture seule en incluant uniquement un accesseur set.

**Liens de référence:**Pour plus d'informations sur les indexeurs, consultez la page Utilisation des indexeurs (Guide de programmation C #) à l'adresse<https://aka.ms/moc-20483c-m3-pg3>.

## ****Démonstration: création et utilisation d'une structure****

Dans cette démonstration, vous allez créer une structure nommée Coffee et ajouter plusieurs propriétés à la structure. Vous allez ensuite créer une instance de la structure Coffee, définir certaines valeurs de propriété et afficher ces valeurs de propriété dans une fenêtre de console.

### **Démonstration étapes**

Vous trouverez les étapes dans la section Démonstration: Création et utilisation d'une structure sur la page suivante:<https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD03_DEMO.md>.

# Leçon 2: Organisation des données en collections

Lorsque vous créez **plusieurs éléments du même type**, qu'il s'agisse **d'entiers**, de **chaînes** ou d'un **type personnalisé tel que Coffee**, vous avez besoin d'un moyen de gérer les **éléments** **en tant qu'ensemble**. Vous devez être en mesure de compter le **nombre** **d'éléments** dans l'ensemble, **d'ajouter** ou de **supprimer** des éléments de l'ensemble et de **parcourir l'ensemble un élément à la fois**. Pour ce faire, vous pouvez utiliser une **collection**.

Les collections sont un outil essentiel pour gérer plusieurs éléments. Ils sont également au cœur du développement d'applications graphiques. Les contrôles tels que les zones de liste déroulante et les menus sont généralement liés aux données des collections.

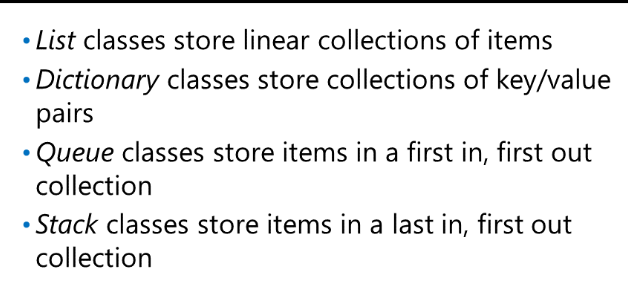
Dans cette leçon, vous apprendrez à utiliser diverses classes de collection dans Visual C #.

## ****Objectifs de la leçon****

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Choisissez des collections appropriées pour différents scénarios. |
| • | Gérez les éléments d'une collection. |
| • | Utilisez la syntaxe LINQ (Language Integrated Query) pour interroger une collection. |

## ****Choisir les collections****



Toutes les classes de collection partagent diverses caractéristiques communes. Pour gérer une collection d'éléments, vous devez être capable de:

|  |  |
| --- | --- |
| • | **Ajoutez** des éléments à la collection. |
| • | **Supprimez** des éléments de la collection. |
| • | **Récupérez** des éléments spécifiques de la collection. |
| • | **Comptez** le nombre d'articles de la collection. |
| • | **Parcourez** les éléments de la collection, un élément à la fois. |

Chaque classe de collection dans Visual C # fournit des méthodes et des propriétés qui prennent en charge ces opérations principales. Au-delà de ces opérations, cependant, vous souhaiterez gérer les collections de différentes manières en fonction des besoins spécifiques de votre application. Les classes de collection dans Visual C # appartiennent aux grandes catégories suivantes:

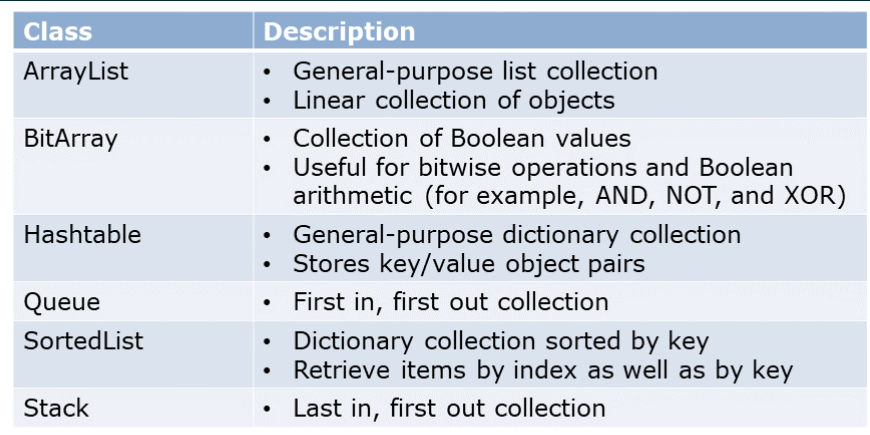
|  |  |
| --- | --- |
| • | les classes **List** stockent des collections linéaires d'éléments. Vous pouvez considérer une classe de liste comme un tableau unidimensionnel qui se développe dynamiquement à mesure que vous ajoutez des éléments. Par exemple, vous pouvez utiliser une classe de liste pour maintenir une liste des boissons disponibles dans votre café. |
| • | Les classes **Dictionnary** stockent une **collection de paires clé / valeur.** **Chaque élément** de la collection se compose de **deux objets**: la **clé** et la **valeur**.  La valeur est l'objet que vous souhaitez stocker et récupérer, et la clé est l'objet que vous utilisez pour indexer et rechercher la valeur.  Dans la plupart des classes de dictionnaire, la clé doit être unique, tandis que les valeurs en double sont parfaitement acceptables. Par exemple, vous pouvez utiliser une classe de dictionnaire pour maintenir une liste de recettes de café. La clé contiendrait le nom unique du café et la valeur contiendrait les ingrédients et les instructions pour préparer le café. |
| • | Les classes **Queue** File d'attente représentent une collection d'objets **premier entré, premier sorti**. Les éléments sont extraits de la collection dans le même ordre qu'ils ont été ajoutés. Par exemple, vous pouvez utiliser une classe de file d'attente pour traiter les commandes dans un café afin de vous assurer que les clients reçoivent leurs boissons à leur tour. |
| • | Les classes **Stack** Empiler représentent une collection d'objets **dernier entré, premier sorti**. L'élément que vous avez ajouté en dernier à la collection est le premier élément que vous récupérez. Par exemple, vous pouvez utiliser une classe de pile pour déterminer les 10 visiteurs les plus récents de votre café. |

Lorsque vous choisissez une classe de collection intégrée pour un scénario spécifique, posez-vous les questions suivantes:

|  |  |
| --- | --- |
| • | Avez-vous besoin d'une liste, d'un dictionnaire, d'une pile ou d'une file d'attente? |
| • | Aurez-vous besoin de trier la collection? |
| • | Quelle est la taille de la collection que vous attendez? |
| • | Si vous utilisez une classe de dictionnaire, aurez-vous besoin de récupérer des éléments par index ainsi que par clé? |
| • | Votre collection se compose-t-elle uniquement de chaînes? |

Si vous pouvez répondre à toutes ces questions, vous pourrez sélectionner la classe de collection Visual C # qui répond le mieux à vos besoins.

## ****Classes de collecte standard****

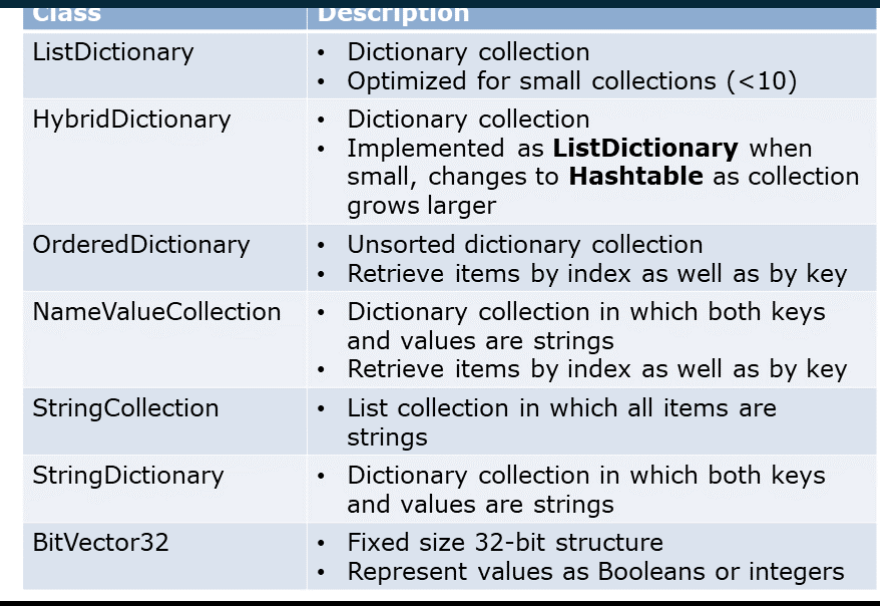


**L'espace de noms System.Collections** fournit une gamme de collections à usage général qui comprend des listes, des dictionnaires, des files d'attente et des piles. Le tableau suivant présente les classes de collection les plus importantes dans l'espace de noms System.Collections:

| **Classer** | **La description** |
| --- | --- |
| **ArrayList** | **ArrayList** est une **liste** à usage **général** qui stocke une collection linéaire d'objets. ArrayList comprend des méthodes et des propriétés qui vous permettent d'ajouter des éléments, de supprimer des éléments, de compter le nombre d'éléments dans la collection et de trier la collection. |
| **BitArray** | BitArray est une classe de liste qui représente une collection de bits sous forme de valeurs booléennes. Le BitArray est le plus couramment utilisé pour les opérations au niveau du bit et l'arithmétique booléenne, et comprend des méthodes pour effectuer des opérations booléennes courantes telles que AND, NOT et XOR. |
| **Hashtable** | La classe **Hashtable** est une classe de **dictionnaire** à usage **général** qui stocke une **collection de paires clé / valeur**. La table de hachage comprend des méthodes et des propriétés qui vous permettent de récupérer des éléments par clé, d'ajouter des éléments, de supprimer des éléments et de rechercher des clés et des valeurs particulières dans la collection. |
| **Queue** | La classe **Queue** est une collection d'objets **premier entré, dernier sorti**. La file d'attente comprend des méthodes pour ajouter des **objets à l'arrière** de la file d'attente (**Enqueue**) et récupérer des **objets à l'avant** de la file d'attente (**Dequeue**). |
| **SortedList** | La classe **SortedList** stocke une collection **de paires clé / valeur triées par clé**. En plus des fonctionnalités fournies par la classe **Hashtable**, la SortedList vous permet de **récupérer** des **éléments** soit par **clé**, soit par **index**. |
| **Stack** | La classe **Stack** est une collection **d'objets premier entré, premier sorti**. La pile inclut des méthodes pour afficher **l'élément supérieur** de la collection sans le supprimer (**Peek**), **ajouter un élément au sommet de la pile (Push**), et **supprimer et renvoyer l'élément en haut de la pile (Pop).** |

**Liens de référence:**Pour plus d'informations sur les classes répertoriées dans le tableau précédent, consultez la page Espace de noms System.Collections à l'adresse<https://aka.ms/moc-20483c-m3-pg4>.

## ****Classes de collection spécialisées****

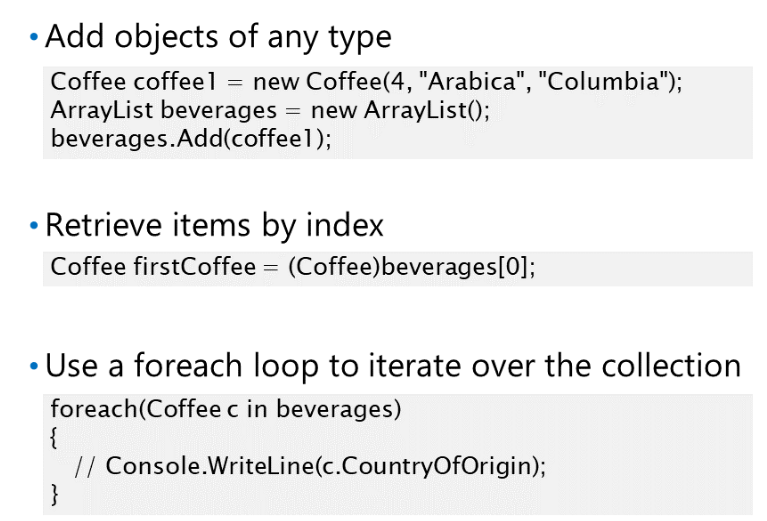


L'espace de noms System.Collections.Specialized fournit des classes de collection adaptées à des exigences plus spécialisées, telles que des collections de dictionnaires spécialisés et des collections de chaînes fortement typées. Le tableau suivant présente les classes de collection les plus importantes dans l'espace de noms System.Collections.Specialized:

| **Classer** | **La description** |
| --- | --- |
| **ListDictionary** | ListDictionary est une classe de dictionnaire optimisée pour les petites collections. En règle générale, si votre collection comprend 10 éléments ou moins, utilisez un ListDictionary. Si votre collection est plus grande, utilisez une table de hachage. |
| **HybridDictionary** | HybridDictionary est une classe de dictionnaire que vous pouvez utiliser lorsque vous ne pouvez pas estimer la taille de la collection. HybridDictionary utilise une implémentation ListDictionary lorsque la taille de la collection est petite et bascule vers une implémentation Hashtable lorsque la taille de la collection augmente. |
| **OrderedDictionary** | OrderedDictionary est une classe de dictionnaire indexée qui vous permet de récupérer des éléments par clé ou par index. Notez que contrairement à la classe SortedList, les éléments d'un OrderedDictionary ne sont pas triés par clé. |
| **NameValueCollection** | NameValueCollection est une classe de dictionnaire indexée dans laquelle la clé et la valeur sont des chaînes. NameValueCollection lèvera une erreur si vous essayez de définir une clé ou une valeur sur autre chose qu'une chaîne. Vous pouvez récupérer des éléments par clé ou par index. |
| **StringCollection** | StringCollection est une classe de liste dans laquelle chaque élément de la collection est une chaîne. Utilisez cette classe lorsque vous souhaitez stocker une collection simple et linéaire de chaînes. |
| **StringDictionary** | Le StringDictionary est une classe de dictionnaire dans laquelle la clé et la valeur sont des chaînes. Contrairement à la classe NameValueCollection, vous ne pouvez pas récupérer les éléments d'un StringDictionary par index. |
| **BitVector32** | Le BitVector32 est une structure qui peut représenter une valeur 32 bits à la fois sous la forme d'un tableau de bits et d'une valeur entière. Contrairement à la classe BitArray, qui peut s'étendre indéfiniment, la structure BitVector32 a une taille 32 bits fixe. En conséquence, le BitVector32 est plus efficace que le BitArray pour les petites valeurs. Vous pouvez diviser une instance BitVector32 en sections pour stocker efficacement plusieurs valeurs. |

**Liens de référence:**Pour plus d'informations sur les classes et les structures répertoriées dans le tableau précédent, consultez la page System.Collections.Specialized Namespace à l'adresse<https://aka.ms/moc-20483c-m3-pg5>.

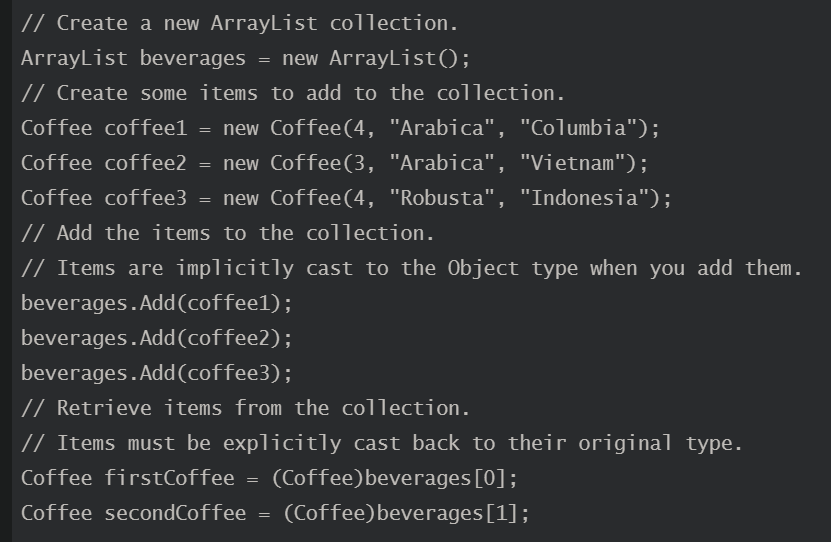
## ****Utilisation des collections de listes****



**ArrayList** stocke les éléments sous la forme d'une collection linéaire d'objets. Vous pouvez **ajouter des objets de n'importe quel type** à une collection ArrayList, mais ArrayList représente chaque élément de la collection comme une instance System.Object. Lorsque vous ajoutez un élément à une collection ArrayList, ArrayList **caste ou convertit implicitement votre élément en type Object**. Lorsque vous récupérez des éléments de la collection, vous devez **explicitement convertir l'objet en son type d'origine.**

L'exemple suivant montre comment ajouter et récupérer des éléments d'une collection ArrayList:

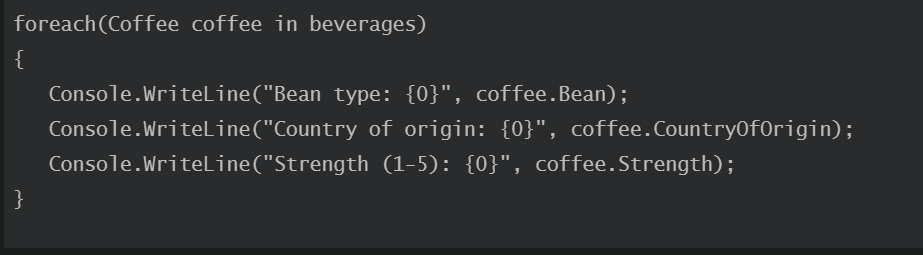
**Ajout et récupération d'éléments à partir d'un ArrayList**



Lorsque vous travaillez avec des collections, l'une de vos tâches de programmation les plus courantes consiste à parcourir la collection. Cela signifie essentiellement que vous récupérez chaque élément de la collection à tour de rôle, généralement pour afficher une liste d'éléments, pour évaluer chaque élément par rapport à certains critères ou pour extraire des valeurs de membre spécifiques de chaque élément. **Pour parcourir une collection, vous utilisez une boucle foreach**. La boucle **foreach** expose tour à tour chaque élément de la collection, en utilisant le nom de variable que vous spécifiez dans la déclaration de boucle.

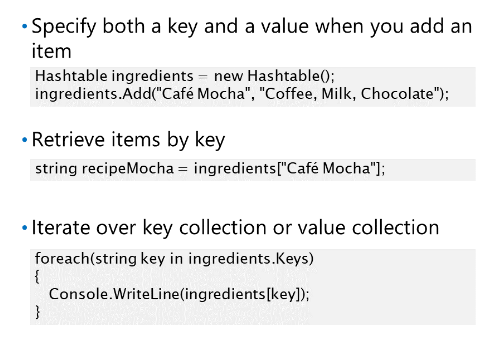
L'exemple suivant montre comment parcourir une collection ArrayList:

**Itération sur une collection de listes**



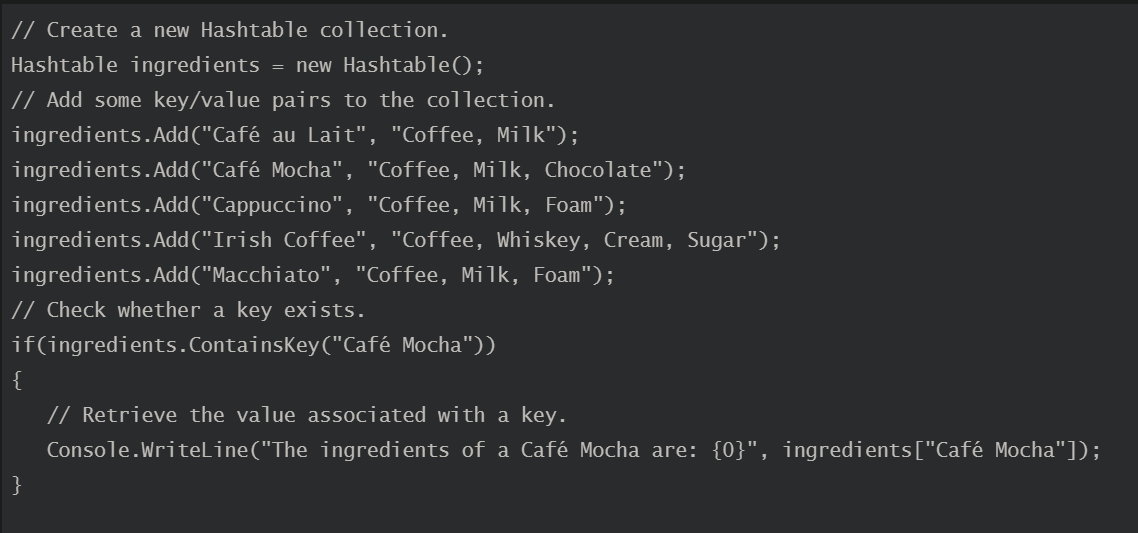
**Liens de référence:**Pour plus d'informations sur la classe ArrayList, consultez la page Classe ArrayList à l'adresse<https://aka.ms/moc-20483c-m3-pg6>.

## ****Utilisation des collections de dictionnaires****



Les classes de **dictionnaire** stockent des **collections de paires clé / valeur**. Une classe de dictionnaire utile est le Hashtable. Lorsque vous ajoutez un élément à une collection Hashtable, vous devez spécifier une clé et une valeur. **La clé et la valeur peuvent être des instances de n'importe quel type**, mais l**a table de hachage convertit implicitement la clé et la valeur en type Object.** **Lorsque vous récupérez des valeurs de la collection, vous devez explicitement convertir l'objet en son type d'origine.**

L'exemple suivant montre comment ajouter et récupérer des éléments d'une collection Hashtable. Dans ce cas, la clé et la valeur sont des chaînes:

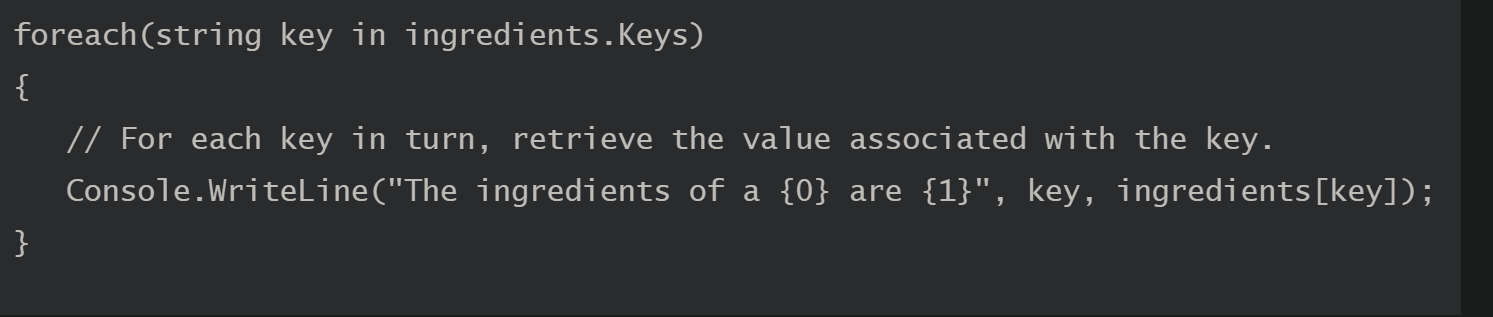


**Ajout et récupération d'éléments à partir d'une table de hachage**

Les classes de dictionnaire, telles que Hashtable, contiennent en fait **deux collections énumérables**: les **Keys** et les **Values**. Vous pouvez **parcourir l'une de ces collections**. Dans la plupart des scénarios, cependant, vous êtes susceptible de parcourir la collection de clés, par exemple pour récupérer la valeur associée à chaque clé à son tour.

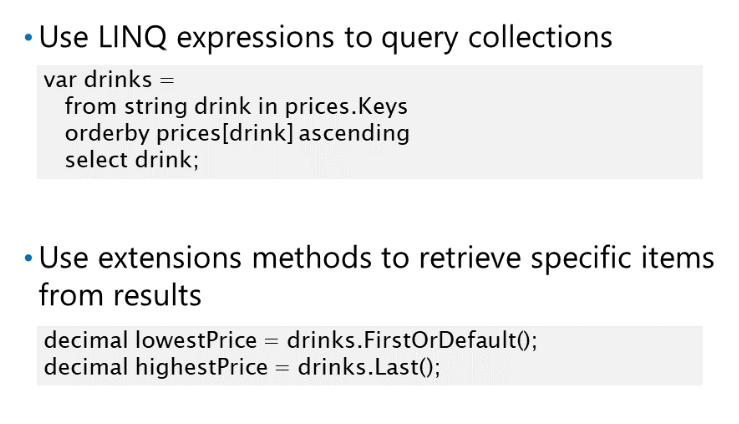
L'exemple suivant montre comment parcourir les Keys d'une collection Hashtable et récupérer la valeur associée à chaque clé:

**Itération sur une collection de dictionnaires**



**Liens de référence:**Pour plus d'informations sur la classe Hashtable, consultez la page Classe Hashtable à l'adresse<https://aka.ms/moc-20483c-m3-pg7>.

## ****Interroger une collection****



**LINQ** est une **technologie de requête** intégrée aux langages .NET tels que Visual C #. **LINQ vous permet d'utiliser une syntaxe de requête déclarative normalisée pour interroger des données à partir d'un large éventail de sources de données**, telles que des **collections** .NET, **des bases de données SQL Server**, des ensembles de **données ADO.NET** et des documents XML. Une requête LINQ de base utilise la syntaxe suivante:

**from** <variable names> **in** <data source>

**where** <selection criteria>

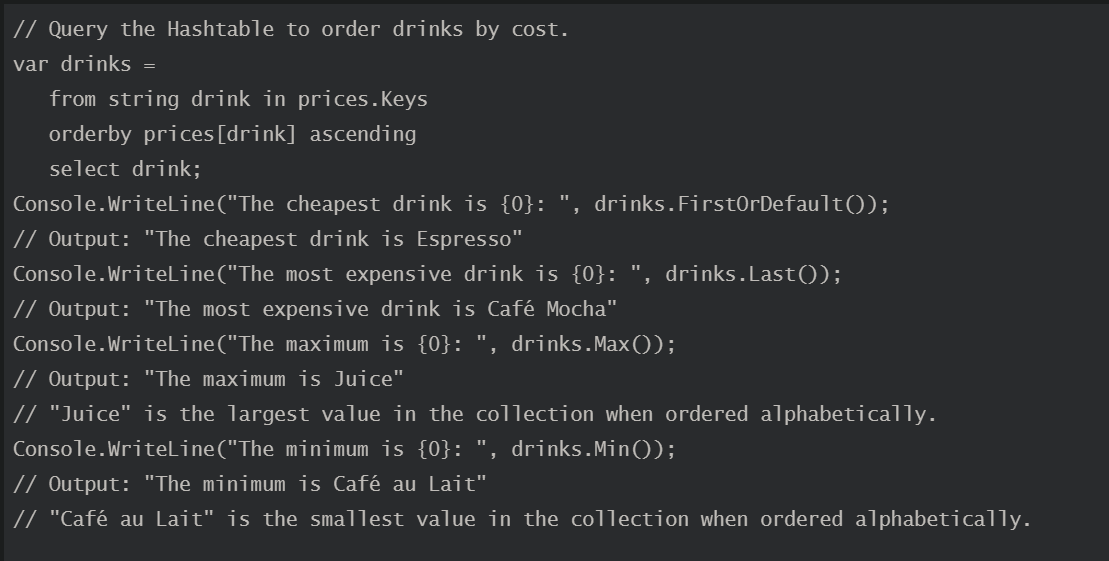
**orderby** <result ordering criteria>

**select** <variable names>

Par exemple, supposons que vous utilisiez une table de hachage pour gérer une liste de prix pour les boissons chez Fourth Coffee. Vous pouvez utiliser une expression LINQ pour récupérer toutes les boissons qui répondent à certains critères de prix, telles que les boissons qui coûtent moins de 2 $.

L'exemple suivant montre comment utiliser une expression LINQ pour interroger une table de hachage:

**Utilisation de LINQ pour interroger une collection**



**Noter:**L'ajout du suffixe m ou M à un nombre indique que le nombre doit être traité comme un type décimal.

En plus de cette syntaxe de requête de base, vous pouvez appeler diverses méthodes sur les résultats de votre requête. Par exemple:

|  |  |
| --- | --- |
| • | Appelez la méthode **FirstOrDefault** pour obtenir le premier élément de la collection de résultats ou une valeur par défaut si la collection ne contient aucun résultat. Cette méthode est utile si vous avez ordonné les résultats de votre requête. |
| • | Appelez la méthode **Last** pour obtenir le dernier élément de la collection de résultats. Cette méthode est utile si vous avez ordonné les résultats de votre requête. |
| • | Appelez la méthode **Max** pour rechercher le plus grand élément de la collection de résultats. |
| • | Appelez la méthode **Min** pour rechercher le plus petit élément de la collection de résultats. |

**Noter:**La plupart des types intégrés fournissent des méthodes qui vous permettent de comparer une instance à une autre pour déterminer laquelle est considérée comme plus grande ou plus petite. Les méthodes Max et Min s'appuient sur ces méthodes pour trouver les éléments les plus grands ou les plus petits. Si votre collection contient des types numériques, ces méthodes renverront respectivement les valeurs les plus élevées et les plus faibles. Si votre collection contient des chaînes, les membres sont comparés par ordre alphabétique. Par exemple, "Z" est considéré comme supérieur à "A".

Si votre **collection** contient des **types personnalisés**, les méthodes Max et Min utiliseront la **logique de comparaison créée par le développeur de types**.

Par exemple, si vous avez trié vos résultats par **coût croissant**, le premier élément de la collection de résultats sera **le moins cher** et le dernier élément de la collection de résultats **sera le plus cher**. En tant que tel, vous pouvez utiliser les méthodes FirstOrDefault et Last pour trouver respectivement les boissons les moins chères et les plus chères.

L'exemple suivant montre comment récupérer les éléments les plus petits et les plus grands d'une collection en fonction des critères de tri dans l'expression LINQ:

**Utilisation des méthodes FirstOrDefault et Last**

// Query the Hashtable to order drinks by cost.

var drinks =

from string drink in prices.Keys

orderby prices[drink] ascending

select drink;

Console.WriteLine("The cheapest drink is {0}: ", drinks.FirstOrDefault());

// Output: "The cheapest drink is Espresso"

Console.WriteLine("The most expensive drink is {0}: ", drinks.Last());

// Output: "The most expensive drink is Café Mocha"

Console.WriteLine("The maximum is {0}: ", drinks.Max());

// Output: "The maximum is Juice"

// "Juice" is the largest value in the collection when ordered alphabetically.

Console.WriteLine("The minimum is {0}: ", drinks.Min());

// Output: "The minimum is Café au Lait"

// "Café au Lait" is the smallest value in the collection when ordered alphabetically.

**Noter:**Le type de retour d'une expression LINQ est IEnumerable <T>, où T est le type des éléments de la collection. IEnumerable <T> est un exemple de type générique. Les méthodes que vous utilisez sur un ensemble de résultats, telles que FirstOrDefault, Last, Max et Min, sont des méthodes d'extension. Les types génériques et les méthodes d'extension sont abordés plus loin dans ce cours.

**Liens de référence:**Pour plus d'informations sur l'utilisation de LINQ pour interroger des collections, consultez la page Expressions de requête LINQ (Guide de programmation C #) à l'adresse<https://aka.ms/moc-20483c-m3-pg8>.

# Leçon 3: Gestion des événements

Les événements sont des mécanismes qui permettent aux objets d'avertir d'autres objets lorsque quelque chose se produit. Par exemple, les contrôles sur une page Web ou dans une interface utilisateur WPF génèrent des événements lorsqu'un utilisateur interagit avec le contrôle, par exemple en cliquant sur un bouton. Vous pouvez créer du code qui s'abonne à ces événements et prend des mesures en réponse à un événement.

Sans événements, votre code aurait besoin de lire en permanence les valeurs de contrôle pour rechercher tout changement d'état nécessitant une action. Ce serait une manière très inefficace de développer une application. Dans cette leçon, vous apprendrez à créer ou à déclencher des événements et à vous abonner à des événements.

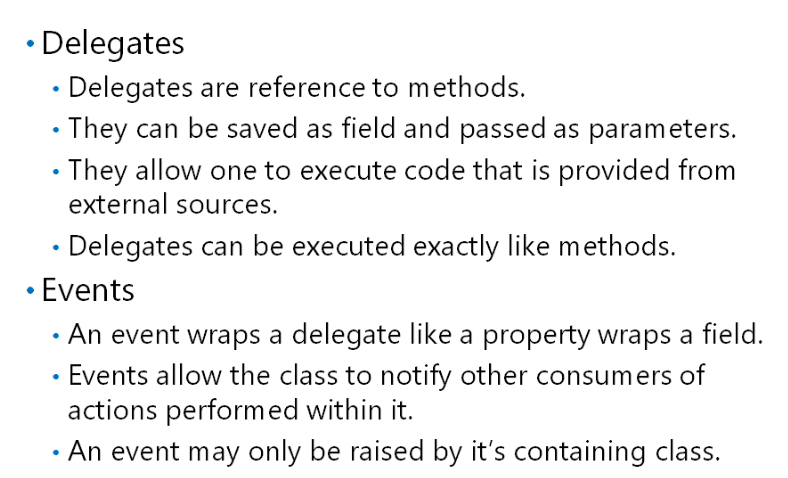
Dans cette leçon, vous apprendrez à gérer les événements.

## **Objectifs de la leçon**

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Créez des événements et des délégués. |
| • | Soulevez des événements. |
| • | Abonnez-vous aux événements et désabonnez-vous des événements. |

## ****Délégués et événements - Introduction****



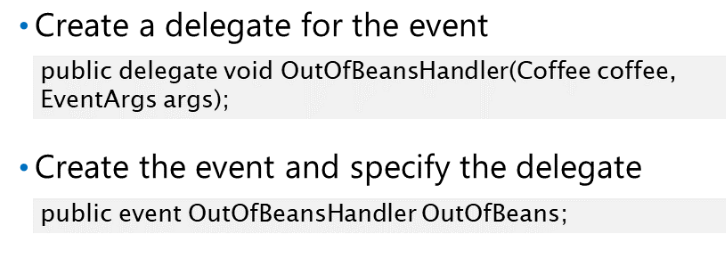
Il existe de nombreux cas lors de la construction d'une application, dans lesquels vous souhaiterez effectuer une action inconnue dans votre classe ou votre contexte actuel, et cette action sera fournie à partir d'une source externe. Par exemple, disons qu'il existe une méthode qui écrit du texte dans une sortie. Vous pouvez laisser l'utilisateur de cette méthode configurer la sortie à l'avance: définir la police, la taille du texte, etc. Peut-être que l'utilisateur voudra peut-être exécuter une autre logique complexe avant d'écrire le texte. Votre méthode aura besoin d'un moyen de recevoir et d'exécuter ce code fourni.

C'est le problème que résolvent les délégués. Un délégué peut être considéré comme une référence à une méthode. Vous pouvez l'enregistrer en tant que champ ou variable et le transmettre comme n'importe quel autre type de référence. La seule différence est que là où les champs de type enregistrent les données, un délégué enregistre la logique. En tant que tel, un délégué peut être appelé exactement comme une méthode. Il peut accepter des paramètres et renvoyer un résultat.

Tout comme les champs standards ont des propriétés pour restreindre l'accès et protéger l'utilisation de leur champ sous-jacent, les délégués ont des événements. Les événements encapsulent leur délégué et empêchent quiconque de les élever ou de remplacer leur valeur. Une classe consommatrice ne peut jamais attribuer une valeur à un événement, comme elle le peut un délégué. Il ne peut que s'abonner et se désinscrire de et de l'événement. Seule la classe contenant peut déclencher (invoquer) l'événement.

Bien qu'il soit techniquement facile d'assimiler les événements aux propriétés en fonction de leurs délégués et champs respectifs, ce n'est pas tout à fait vrai. Alors que les propriétés fournissent simplement un accès protégé à leur champ, les événements sont conceptuellement différents des délégués, même s'ils dépendent d'eux. Alors que les délégués sont souvent utilisés pour exécuter la logique fournie au processus de l'extérieur, les événements sont utilisés pour informer d'autres classes ou consommateurs de la classe qu'une certaine action vient de se produire (en d'autres termes - qu'un événement s'est produit). Par conséquent, les événements sont rarement utilisés pour influencer le processus actuel (bien que cela soit possible), mais pour démarrer d'autres processus dans les cibles souscrites.

## ****Création d'événements et de délégués****



Lorsque vous créez un événement dans une structure ou une classe, vous avez besoin d'un moyen d'activer un autre code pour s'abonner à votre événement. Dans Visual C #, vous accomplissez cela en créant un délégué. Un délégué est un type spécial qui définit une signature de méthode; en d'autres termes, le type de retour et les paramètres d'une méthode. Comme son nom l'indique, un délégué se comporte comme un représentant pour les méthodes avec des signatures correspondantes.

Lorsque vous définissez un événement, vous associez un délégué à votre événement. Pour vous abonner à l'événement à partir du code client, vous devez:

|  |  |
| --- | --- |
| • | Créez une méthode avec une signature qui correspond au délégué d'événement. Cette **méthode** est connue sous **le nom de gestionnaire d'événements.** |
| • | **Abonnez**-vous à l'événement en **donnant le nom de votre méthode de gestionnaire d'événements à l'éditeur d'événements**, en d'autres termes, **l'objet qui déclenchera l'événement.** |

Lorsque **l'événement est déclenché**, ***le délégué appelle toutes les méthodes de gestionnaire d'événements qui se sont abonnées à l'événement.***

Supposons que vous créez une structure nommée Coffee.

L'une des responsabilités de cette structure est de **suivre le niveau de stock pour chaque instance de café.**

***Lorsque le niveau de stock descend en dessous d'un certain point, vous souhaiterez peut-être déclencher un événement pour avertir un système de commande que vous manquez de beans.***

La première chose à faire est de définir un délégué. Pour définir un délégué, vous utilisez le mot-clé **delegate**. Un délégué comprend deux paramètres:

|  |  |
| --- | --- |
| • | Le premier paramètre est **l'objet qui a déclenché l'événement - dans ce cas, une instance Coffee.** |
| • | Le deuxième paramètre **est les arguments** d'événement, c'est-à-dire toute autre information que vous souhaitez fournir aux consommateurs. Il doit s'agir ***d'une instance de la classe EventArgs ou d'une instance d'une classe qui dérive d'EventArgs.*** |

Ensuite, vous devez définir l'événement. Pour **définir un événement**, vous utilisez le mot-clé **event**. Vous faites précéder le nom de votre événement du nom du délégué que vous souhaitez associer à votre événement.

L'exemple suivant montre comment définir des délégués et des événements:

**Définition d'un délégué et d'un événement**

public struct Coffee

{

public EventArgs e;

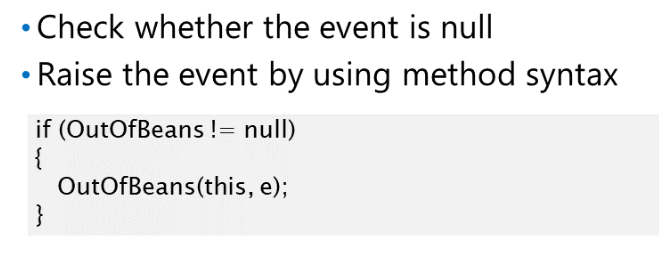
public ***delegate*** void **OutOfBeansHandler**(Coffee coffee, EventArgs args);

public ***event*** **OutOfBeansHandler** OutOfBeans;

}

Dans cet exemple, vous définissez un **événement** nommé **OutOfBeans**. Vous associez un délégué nommé OutOfBeansHandler à votre événement. Le délégué OutOfBeansHandler prend deux paramètres, une instance de Coffee qui représentera l'objet qui a déclenché l'événement et une instance d'EventArgs qui pourrait être utilisée pour fournir plus d'informations sur l'événement.

## ****Élever des événements (Raising Events)****



Après avoir défini un événement et un délégué, vous pouvez écrire du code qui déclenche l'événement lorsque certaines conditions sont remplies. Lorsque vous déclenchez l'événement, le délégué associé à votre événement appellera toutes les méthodes de gestionnaire d'événements qui se sont abonnées à votre événement.

Pour ***déclencher un événement***, vous devez faire deux choses:

|  |  |
| --- | --- |
| 1. | Vérifiez si l'événement est nul. L'événement sera nul si aucun code n'y est actuellement abonné. |
| 2. | Appelez l'événement et fournissez des arguments au délégué. |

Par exemple, supposons qu'une structure Coffee inclut une méthode nommée MakeCoffee. Chaque fois que vous appelez la méthode MakeCoffee, la méthode réduit le niveau de stock de l'instance Coffee. Si le niveau de stock tombe en dessous d'un certain point, la méthode MakeCoffee déclenchera un événement OutOfBeans.

L'exemple suivant montre comment déclencher un événement:

**Organiser un événement**

public struct Coffee

{

// Declare the event and the delegate.

public EventArgs e = null;

public delegate void OutOfBeansHandler(Coffee coffee, EventArgs args);

public event OutOfBeansHandler OutOfBeans;

int currentStockLevel;

int minimumStockLevel;

public void MakeCoffee()

{

// Decrement the stock level.

currentStockLevel--;

// If the stock level drops below the minimum, raise the event.

if (currentStockLevel < minimumStockLevel)

{

// Check whether the event is null.

if (OutOfBeans != null)

{

// Raise the event.

OutOfBeans(this, e);

}

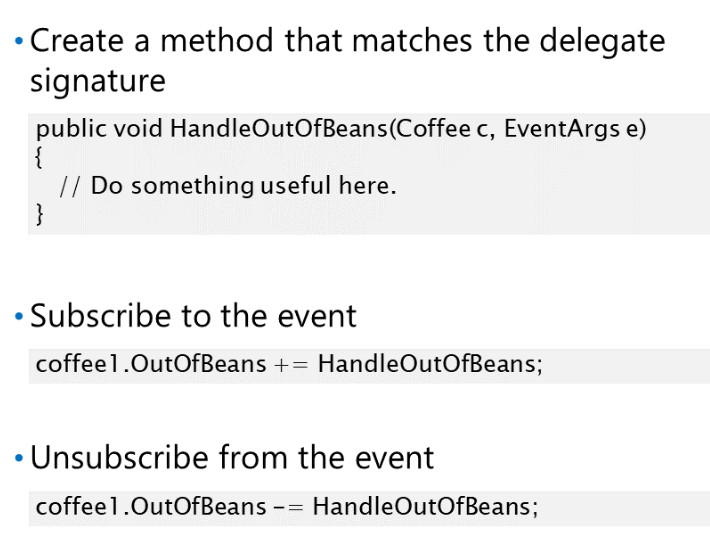
}

}

}

Pour déclencher l'événement, vous utilisez une syntaxe similaire à l'appel d'une méthode. Vous fournissez des arguments pour correspondre aux paramètres requis par le délégué. Le premier argument est l'objet qui a déclenché l'événement. Notez comment le mot-clé this est utilisé pour indiquer l'instance Coffee actuelle. Le deuxième paramètre est l'instance EventArgs, qui peut être null si vous n'avez pas besoin de fournir d'autres informations aux abonnés.

## ****S'abonner à des événements****



Si vous souhaitez gérer un événement dans le code client, vous devez effectuer deux opérations:

|  |  |
| --- | --- |
| • | Créez une méthode avec une signature qui correspond au délégué de l'événement. |
| • | Utilisez l'opérateur d'affectation d'addition (+ =) pour attacher votre méthode de gestionnaire d'événements à l'événement. |

Par exemple, supposons que vous ayez créé une instance de la structure Coffee nommée coffee1. Dans votre classe Inventory, vous souhaitez vous abonner aux OutOfBeans qui peuvent être générés par coffee1.

**Noter:**La rubrique précédente montre comment la structure Coffee, l'événement OutOfBeans et le délégué OutOfBeansHandler sont définis.

L'exemple suivant montre comment s'abonner à un événement:

**S'abonner à un événement**

public class Inventory

{

public void HandleOutOfBeans(Coffee sender, EventArgs args)

{

string coffeeBean = sender.Bean;

// Reorder the coffee bean.

}

public void SubscribeToEvent()

{

coffee1.OutOfBeans += HandleOutOfBeans;

}

}

Dans cet exemple, la signature de la méthode HandleOutOfBeans correspond au délégué de l'événement OutOfBeans. Lorsque vous appelez la méthode SubscribeToEvent, la méthode HandleOutOfBeans est ajoutée à la liste des abonnés pour l'événement OutOfBeans sur l'objet coffee1.

Pour vous désabonner d'un événement, vous utilisez l'opérateur d'affectation de soustraction (- =) pour supprimer votre méthode de gestionnaire d'événements de l'événement. Vous pouvez vous désabonner d'un événement auquel vous n'êtes pas abonné en toute sécurité. Dans ce cas, rien ne se passera, aucune exception ne sera lancée.

L'exemple suivant montre comment se désabonner d'un événement:

**Se désabonner d'un événement**

public void UnsubscribeFromEvent ()

{

coffee1.OutOfBeans - = HandleOutOfBeans;

}

Il est important de vous désabonner des événements lorsque vous en avez terminé avec eux ou avec l'objet abonné. Un abonnement à un événement enregistre une référence à l'instance d'objet abonnée. Le fait de ne pas se désabonner peut entraîner une fuite de mémoire.

**Liens de référence:**Pour plus d'informations, consultez la page Comment: s'abonner et se désabonner des événements (Guide de programmation C #) à l'adresse<https://aka.ms/moc-20483c-m3-pg9>.

**Démonstration: utilisation des événements en XAML**

Visual Studio fournit des outils qui facilitent l'utilisation des événements dans les applications WPF. Dans cette démonstration, vous apprendrez à vous abonner aux événements déclenchés par les contrôles WPF.

### **Démonstration étapes**

Vous trouverez les étapes dans la démonstration: Utilisation d'événements en XAML section sur la page suivante: You will find the steps in the Demonstration: Working with Events in XAML section on the following page:<https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD03_DEMO.md>.

## **Démonstration: rédaction de code pour le laboratoire d'application du prototype de grades**

Dans cette démonstration, vous découvrirez les tâches que vous effectuerez dans le laboratoire pour ce module.

### **Démonstration étapes**

Vous trouverez les étapes dans la section Démonstration: Écriture du code pour le laboratoire d'application du prototype de grades sur la page suivante:<https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD03_DEMO.md>.

# **Laboratoire: Rédaction du code pour l'application de prototype Grades**

### **Scénario**

L'École des Beaux-Arts a décidé d'étendre sa demande d'inscription en classe de base pour permettre aux enseignants d'enregistrer les notes que les élèves de leur classe ont obtenues pour chaque matière et pour permettre aux élèves de consulter leurs propres notes. Cette fonctionnalité nécessite la mise en œuvre de la fonctionnalité de connexion aux applications pour authentifier l'utilisateur et déterminer si l'utilisateur est un enseignant ou un étudiant.

Vous décidez de commencer par développer des parties d'un prototype d'application pour tester la preuve de concept et obtenir les commentaires des clients avant de vous lancer dans l'application finale. L'application prototype utilisera des vues WPF de base plutôt que des formulaires séparés pour l'interface utilisateur. Ces vues ont déjà été conçues et vous devez ajouter le code pour naviguer parmi elles.

Vous décidez également de commencer par stocker les informations sur l'utilisateur et la note dans des structures de base et d'utiliser une source de données factice dans l'application pour tester votre fonctionnalité de connexion.

### **Objectifs**

Après avoir terminé cet atelier, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Naviguez parmi les vues. |
| • | Créez et utilisez des collections de structures. |
| • | Gérez les événements. |

##### **Configuration du laboratoire**

Durée estimée: 90 minutes

Vous trouverez les étapes de haut niveau sur la page suivante:<https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD03_LAB_MANUAL.md>.

Vous trouverez les étapes détaillées sur la page suivante:<https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD03_LAK.md>.

### **Exercice 1: Ajout d'une logique de navigation à l'application de prototype de grades**

##### **Scénario**

Dans cet exercice, vous allez ajouter une logique de navigation à l'application GradesPrototype.

Tout d'abord, vous examinerez la fenêtre et les vues de l'application afin de vous familiariser avec la structure existante de l'application. Vous allez définir un gestionnaire d'événements public nommé LogonSuccess qui sera déclenché lorsqu'un utilisateur se connecte avec succès à l'application. Vous allez ajouter un code factice au gestionnaire d'événements Logon\_Click pour stocker le nom d'utilisateur et le rôle de l'utilisateur connecté et déclencher l'événement LogonSuccess. Ensuite, vous ajouterez un balisage au code XAML LogonPage pour connecter le bouton d'ouverture de session au gestionnaire d'événements Logon\_Click. Ensuite, vous ajouterez du code à la méthode GotoLogon pour afficher la vue d'ouverture de session et pour masquer les autres vues. Vous implémenterez la méthode Logon\_Success pour gérer une connexion réussie en affichant les vues connectées, puis vous ajouterez un balisage au code XAML MainWindow pour connecter l'événement LogonSuccess à la méthode Logon\_Success. Vous allez ajouter du code à MainWindow pour déterminer si l'utilisateur est un enseignant ou un étudiant, afficher son nom dans l'application et afficher la vue StudentsPage pour les enseignants ou la vue StudentProfile pour les étudiants. Vous allez ensuite ajouter du code à la vue StudentsPage qui détecte le nom d'un étudiant sur lequel l'utilisateur clique et déclenche l'événement StudentSelected pour cet étudiant et affiche son profil d'étudiant. Enfin, vous exécuterez l'application et vérifierez que les vues appropriées sont affichées pour les étudiants et les enseignants après une connexion réussie. Vous allez ensuite ajouter du code à la vue StudentsPage qui détecte le nom d'un étudiant sur lequel l'utilisateur clique et déclenche l'événement StudentSelected pour cet étudiant et affiche son profil d'étudiant. Enfin, vous exécuterez l'application et vérifierez que les vues appropriées sont affichées pour les étudiants et les enseignants après une connexion réussie. Vous allez ensuite ajouter du code à la vue StudentsPage qui détecte le nom d'un étudiant sur lequel l'utilisateur clique et déclenche l'événement StudentSelected pour cet étudiant et affiche son profil d'étudiant. Enfin, vous exécuterez l'application et vérifierez que les vues appropriées sont affichées pour les étudiants et les enseignants après une connexion réussie.

**Résultat**: Après avoir terminé cet exercice, vous devez avoir mis à jour l'application Grades Prototype pour répondre aux événements utilisateur et vous déplacer de manière appropriée parmi les vues de l'application.

### **Exercice 2: Création de types de données pour stocker les informations sur les utilisateurs et les notes**

##### **Scénario**

Dans cet exercice, vous définirez des structures de base qui contiendront les informations relatives à l'enseignant, à l'étudiant et à la note de l'application. Vous examinerez ensuite la source de données factice que l'application utilise pour remplir les collections dans cet exercice.

**Résultat**: Une fois cet exercice terminé, l'application contiendra des structures pour les types d'enseignant, d'élève et de note.

### **Exercice 3: Affichage des informations sur l'utilisateur et la note**

##### **Scénario**

Dans cet exercice, vous allez d'abord définir un gestionnaire d'événements public nommé LogonFailed qui sera déclenché lorsqu'un utilisateur ne parvient pas à se connecter. Vous allez ajouter du code au gestionnaire d'événements Logon\_Click pour valider le nom d'utilisateur et le mot de passe entrés par l'utilisateur par rapport à la collection Users dans la fenêtre MainWindow. Si l'utilisateur est un enseignant ou un étudiant, vous stockerez ses détails dans le contexte global, puis déclencherez l'événement LogonSuccess, mais si l'utilisateur n'est pas validé, vous déclencherez l'événement LogonFailed. Vous gérerez l'échec de la connexion dans la méthode Logon\_Failed pour afficher un message à l'utilisateur, puis vous ajouterez un balisage au code XAML MainWindow pour connecter l'événement LogonFailed à la méthode Logon\_Failed. Vous allez ajouter du code à la vue StudentsPage pour afficher les élèves de l'enseignant actuel, et pour afficher les détails d'un étudiant lorsque l'utilisateur clique sur son nom. Vous utiliserez ensuite la liaison de données pour afficher les détails et les notes de l'élève actuel dans la vue StudentProfile, et pour afficher uniquement le bouton Retour si l'utilisateur est un enseignant. Enfin, vous exécuterez l'application et vérifierez que seuls les utilisateurs valides peuvent se connecter et que les utilisateurs valides ne peuvent voir que les données appropriées à leur rôle.

**Résultat**: Une fois cet exercice terminé, seuls les utilisateurs valides pourront se connecter à l'application et ne verront que les données correspondant à leur rôle.

# **Revue du module et points à retenir**

Dans ce module, vous avez appris à implémenter des structures et des énumérations, à organiser les données en collections et à utiliser des événements et des délégués.

### **Question (s) de révision**