Module 6

Créer de Nouveaux Types

Sommaire

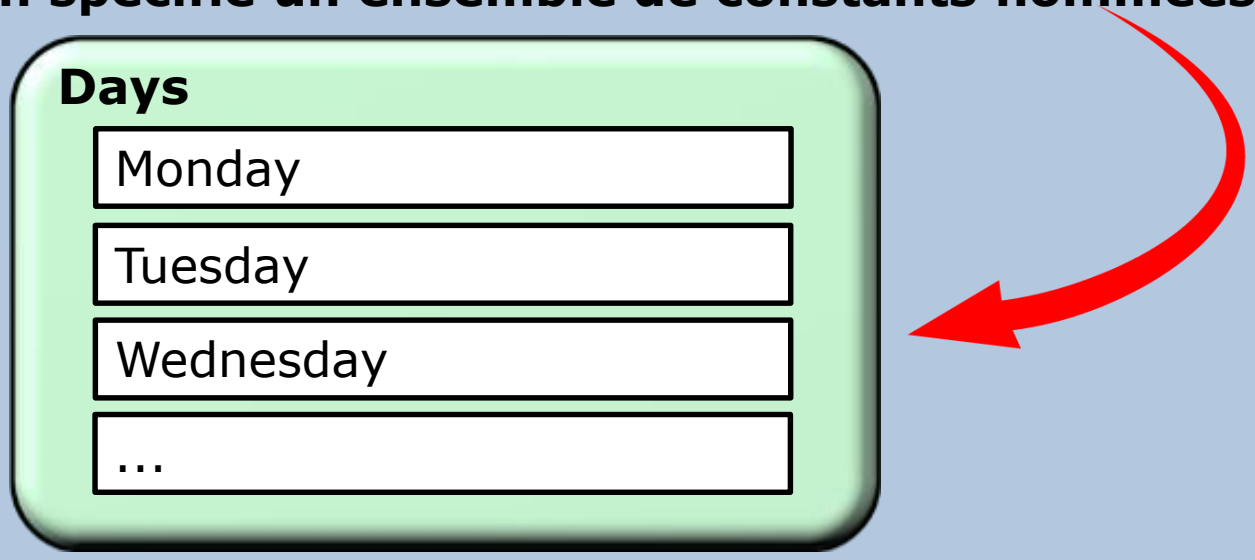
- Créer et Utiliser des Enumérations
- Créer et Utiliser des Classes
- Créer et Utiliser des Structures
- Comparer Types par Références et par Valeur

Leçon 1: Créer et Utiliser des Enumérations

- Qu’est-ce que les Enumerations?
- Créer de Nouveaux Types Enum
- Initialiser et Assigner des Variables aux Enums

Qu’est-ce que les Enumerations?

Une énumeration spécifie un ensemble de constants nommées



Benefices:

- Le code est plus facile à maintenir car vous affectez uniquement les valeurs attendues à vos variables
- Le code est plus facile à lire parce que vous assigner des noms facilement identifiables à vos valeurs
- Le code est plus facile à écrire parce que l’IntelliSense affiche une liste des valeurs possibles que vous pouvez utiliser

Créer de Nouveaux Types Enum

Définir des énumérations avec le mot clé **enum**

```
enum Name : DataType { Value1, Value2 . . . };
```

Vous pouvez définir des énumérations dans les espaces de noms et classes, mais pas dans des méthodes

Les valeurs des énumérations commencent à 0 par défaut, sauf si expressément indiqué

```
namespace Fabrikam.CustomEnumerations
{
    enum Days : int { Monday = 1, Tuesday = 2, Wednesday = 3 };
    ...
}
```

Initialiser et Assigner des Variables aux Enums

Instancier des variables de type énumération de la même manière que tout autre type

```
[EnumType] variableName = [EnumValue]
```

Vous pouvez uniquement assigner une valeur définie dans la définition de l'énum

```
enum Days : int
{
    Monday = 1, Tuesday = 2, Wednesday = 3, Thursday = 4,
    Friday = 5, Saturday = 6, Sunday = 7
};
static void Main(string[] args)
{
    Days myDayOff = Days.Sunday;
}
```

Assigner la valeur **Sunday**

Vous pouvez effectuer des opérations simples sur une énumération

```
for (Days dayOfWeek = Days.Monday; dayOfWeek <= Days.Sunday;
    dayOfWeek++)
{
    ...
}
```

Lesson 2: Créer et Utiliser des Classes

- Qu’est-ce qu’une Classe?
- Ajouter des Membres aux Classes
- Définir des Constructeurs et Initialiser un Objet
- Créer des Objets
- Accéder aux Membres de la Classe
- Utiliser des Classes et Méthodes Partielles

Qu’est-ce qu’une Classe?

Une classe est un plan à partir duquel vous pouvez créer des objets

Une classe définit les caractéristiques d’un objet

```
class House
{
    ...
}
```

Définition de la classe avec les mot clé **class**

Un objet est une instance d’une classe

myHouse

yourHouse

Ajouter des Membres aux Classes

Les membres définissent les données et le comportement de la classe

```
public class Residence
{
    public ResidenceType type;
    public int numberOfBedrooms;
    public bool hasGarage;
    public bool hasGarden;

    public int CalculateSalePrice()
    {
        // Code to calculate the sale value of
        // the residence.
    }

    public int CalculateRebuildingCost()
    {
        // Code to calculate the rebuilding costs
        // of the residence.
    }
}
```

Champs

Méthodes

Définir des Constructeurs et Initialiser un Objet

Un constructeur est une méthode spéciale que le runtime appelle implicitement

```
public class Residence
{
    public Residence(ResidenceType type, int numberOfBedrooms)
    {
    }
    public Residence(ResidenceType type, int numberOfBedrooms,
        bool hasGarage)
    {
    }
    public Residence(ResidenceType type, int numberOfBedrooms,
        bool hasGarage, bool hasGarden)
    {
    }
}
```

3 constructeurs

Si vous n'initialisez pas un champ dans une classe, il est assigné à sa valeur par défaut

Créer des Objets

Les objets sont initialement non assignées

Avant de pouvoir utiliser une classe, vous devez créer une instance de cette classe

Vous pouvez créer une nouvelle instance d'une classe à l'aide de l'opérateur new

L'opérateur **new** fait 2 choses:

- Appelle le CLR pour allouer de la mémoire pour l'objet
- Appelle un constructeur pour initialiser l'objet

```
// Create a Flat with 2 bedrooms.
Residence myFlat = new Residence(ResidenceType.Flat, 2);

// Create a House with 3 bedrooms and a garage
Residence myHouse = new Residence(ResidenceType.House, 3, true);

// Create a Bungalow with 2 bedrooms, a garage, and a garden
Residence myBungalow =
   new Residence(ResidenceType.Bungalow, 2, true, true);
```

Accéder aux Membres de la Classe

Accéder aux membres de l'objet en utilisant le nom de l'instance suivi d'un point

```
InstanceName.MemberName
```

Exemple

```
// Create a 3 bedroom house.
Residence myHouse = new Residence(ResidenceType.House, 3);

// Indicate that the residence has a garden.
myHouse.hasGarden = true;

// Calculate the market value.
int salePrice = myHouse.CalculateSalePrice();

// Get the rebuilding costs.
int rebuildCost = myHouse.CalculateRebuildingCost();
```

Utiliser des Classes et Méthodes Partielles

Fractionner une définition de classe dans plusieurs fichiers sources

Préfixer Les déclarations de classe et de méthode avec le mot clé partial

```
public partial class Residence
{
    partial void SaleResidence(string name);
}
```

Définition de la méthode **SaleResidence**

```
public partial class Residence
{
    partial void SaleResidence(string name)
    {
        //Logic goes here.
    }
}
```

Implémentation de la méthode **SaleResidence**

Toutes les parties de la classe et méthode partielle doivent être disponibles lors de la compilation

Compiler en une seule entité

Leçon 3: Créer et Utiliser des Structures

- Qu’est-ce que les Structures?
- Définir et Utiliser une Structure
- Initialiser une Structure

Qu’est-ce que les Structures?

Toutes les structures sont des types par valeur

System.Byte	System.Int64	System.Decimal
System.Int16	System.Single	System.Boolean
System.Int32	System.Double	System.Char

Les données dans les structures sont stockées dans la pile

Utilisez les structures pour modéliser des entités qui contiennent peu de données

Les structures peuvent contenir des champs et des méthodes comme les classes

```
int x = 99;
string xAsString = x.ToString();
```

Définir et Utiliser une Structure

Utilisez les mot clé **struct** pour declarer une structure

Définissez les membres de la même façon que dans une classe

```
struct Currency
{
    public string currencyCode;
    public string currencySymbol;
    public int fractionDigits
}
```

Les Structures sont automaticament allouées dans la pile mémoire

```
Currency unitedStatesCurrency;

unitedStatesCurrency.currencyCode = "USD";
unitedStatesCurrency.currencySymbol = "$";
unitedStatesCurrency.fractionDigits = 2;
```

Vous n'avez pas besoin d'utilizer l'opérateur **new** quand vous créez une instance d'une structure

Initialiser une Structure

L'opérateur **new** initialise uniquement les champs de la structure

```
struct Currency
{
    public string currencyCode;
    public string currencySymbol;
    public int fractionDigits
    public Currency(string code, string symbol)
    {
        this.currencyCode = code;
        this.currencySymbol = symbol;
        this.fractionDigits = 2;
    }
};
...
Currency unitedKindgdomCurrency = new Currency("GBP", "£");
```

Toujours utiliser un constructeur pour garantir que la structure et les champs sont initialisés

Leçon 4: Comparer Types par Références et par Valeur

- Comparer les Types de référence et les Types par valeur
- Passage par référence d’un Type par valeur à une méthode
- Boxing et Unboxing
- Types Nullables

Comparer les Types de référence et les Types par valeur

Types par Reference (référence dans le tas, valeur dans la pile)

- Quand vous assignez une référence, vous faites simplement référence à un objet en mémoire
- Si vous affectez la même référence à deux variables différentes, les deux variables font référence au même objet
- Si vous modifiez les données dans l'objet, les changements se refléteront dans toutes les variables qui font référence à cet objet

Types par Valeur (valeur dans la pile)

- Quand vous copiez un type par valeur, les variables ne se réfèrent pas au même objet
- Si vous modifiez les données dans une variable, les changements ne figurera pas dans les autres copies

Passage par référence d'un Type par valeur à une méthode

```
static void Main(string[] args)
{
    int myInt = 1005;
    ChangeInput(myInt);
}

static void ChangeInput(int input)
{
    input = 29910;
}
```

myInt still equals 1005

```
static void Main(string[] args)
{
    int myInt = 1005;
    ChangeInput(ref myInt);
}

static void ChangeInput(ref int input)
{
    input = 29910;
}
```

myInt now equals 29910

Boxing et Unboxing

Boxing (type par valeur vers type par référence)

1. Le CLR alloue un peu de mémoire dans le tas
2. Le CLR copie la valeur depuis la pile vers la nouvelle zone mémoire

```
Currency myCurrency = new Currency();  // Value type
object o = myCurrency; // Box the value type into a reference
```

Unboxing (type par référence vers type par valeur)

1. The CLR vérifie le type de la variable boxée
2. Siles types correspondent, le CLR copie la valuer dans la pile

```
Currency myCurrency = new Currency(...);
object o = myCurrency; // boxing
...
Currency anotherCurrency = (Currency)o; // compiles okay
```

Types Nullables

Vous pouvez définir une variable de référence à null pour indiquer qu'elle n'a pas été initialisée

La valeur nulle est elle-même une référence, et il n'y a aucune valeur correspondante pour les types par valeur pour indiquer qu'ils ne sont pas initialisés

```
Currency myCurrency = null; // illegal
```

Vous utilisez ? pour indiquer qu'un type par valeur est nullable

```
Currency? myCurrency = null; // legal
if (myCurrency == null)
{
}
```

Les types nullables exposent les propriétés HasValue et Value

```
Currency? myCurrency = null;
if (myCurrency.HasValue)
{
  Console.WriteLine(myCurrency.Value);
}
```

Atelier Pratique

- Exercice 1:
- Exercice 2:
- Exercice 3:
- Exercice 4: