Module 9:
Création et destruction d'objets

technofuturTIC

WAVENET
Intelligent IT Solutions

Vue d'ensemble

- Utilisation de constructeurs
- Initialisation des données
- Objets et mémoire
- Gestion des ressources

technofuturTIC

WAVENET
Intelligent IT Solutions

Utilisation de constructeurs

- Création d'objets
- Utilisation du constructeur par défaut
- Redéfinition du constructeur par défaut
- Surcharge Constructeurs

Création d'objets

- Étape 1: Allocation de mémoire
  - Utilisez mot-clé **new** pour allouer de la mémoire dans le heap (tas)
- Étape 2: Initialisation de l'objet en utilisant un constructeur
  - Utilisez le nom de la classe, suivi par des parenthèses

```
Date when = new Date();
```

En utilisant le constructeur par défaut

- Caractéristiques d'un constructeur par défaut
  - L'accès du public
  - Même nom que la classe
  - Aucun type-retour pas même **void**
  - Aucuns arguments
  - Initialise tous les champs **zéro**,**faux** ou **nul**
- Syntaxe de constructeur :

```
class Date {public Date () {...}}
```

technofutur TIC

WAVENET Intelligent IT Solutions

Redéfinition du constructeur par défaut

- Le constructeur par défaut pourrait être inapproprié
  - Si c'est le cas, ne pas l'utiliser, écrivez le votre!

```
class Date
{
    public Date( )
    {
        ccyy = 1970;
        mm = 1;
        dd = 1;
    }
    private int ccyy, mm, dd;
}
```

technofutur TIC

WAVENET Intelligent IT Solutions

Surcharge Constructeurs

- Les constructeurs sont des méthodes et peuvent donc être surchargés
  - Même portée, même nom, différents paramètres
  - Permet aux objets d'être initialisés de différentes façons
- AVERTISSEMENT
  - Si vous écrivez un constructeur d'une classe, le compilateur ne crée pas un constructeur par défaut

```
classe Date
{
    public Date () {...}
    public Date (int année, Int mois, Int jour) {...}
    ...
}
```

technofutur TIC

WAVENET Intelligent IT Solutions

Initialisation des données

- Utilisation des listes d'initialiseurs
- Déclaration de variables et constantes en lecture seule
- Initialisation des champs readonly
- Déclarer un constructeur pour une structure
- Utilisation de constructeurs privés
- Utilisation constructeurs statiques

technofutur TIC

WAVENET Intelligent IT Solutions

Utilisation des listes de initialiseur

- Des constructeurs surchargés pourraient contenir du code en double
  - Factoriser en faisant appel aux autres constructeurs
  - Utilisez le mot-clé **this** dans une liste d'initialisation

```
class Date
{
    ...
    public Date( ) : this(1970, 1, 1) { }
    public Date(int year, int month, int day) { ... }
}
```

technofutur TIC

WAVENET Intelligent IT Solutions

Déclaration de variables et constantes en lecture seule

- Valeur de champ constant est obtenu au moment de la compilation

- Valeur de champ readonly est obtenu au moment de l'exécution

technofutur TIC

WAVENET Intelligent IT Solutions

Initialisation Readonly champs

- Champs en lecture seule doivent être initialisées
  - Implicitement à zéro, **faux** ou **nul**
  - Explicitement à leur déclaration dans un initialiseur de variable
  - Explicitement l'intérieur d'un constructeur d'instance

```
class SourceFile
{
    private readonly ArrayList lines;
}
```

technofutur TIC

WAVENET
Intelligent IT Solutions

Déclarer un constructeur pour une structure

- Le compilateur
  - Génère toujours un constructeur par défaut. Les constructeurs par défaut initialisent automatiquement tous les champs à zéro.
- Le développeur
  - Peut déclarer des constructeurs avec un ou plusieurs arguments. Les constructeurs déclarés n'initialisent pas automatiquement les champs à zéro.

technofutur TIC

WAVENET
Intelligent IT Solutions

Utilisation de constructeurs privés

- Un constructeur privé empêche les objets non désirés d'être créé
  - Les méthodes d'instance ne peuvent pas être appelés
  - Les méthodes statiques peuvent être appelées
  - Un bon moyen de mettre en œuvre des fonctions de procédure

```
public class Math
{
    public static double Cos(double x) { ... }
    public static double Sin(double x) { ... }
    private Math( ) { }
}
```


Utilisation Constructeurs statiques

- But
  - Appelé par le “Class Loader” au moment de l'exécution
  - Peut être utilisé pour initialiser les champs statiques
  - Appelé avant les constructeur d'instance
- Restrictions
  - Ne peut pas être appelé directement
  - Aucun modificateur d'accès
  - Doit être sans paramètre

Objets et mémoire

- Cycle de vie d'un objet
- Objets et portée
- Garbage Collection

Cycle de vie d'un objet

1. Création d'objets
   - Vous allouez de la mémoire à l'aide de **new**
   - Vous **initialisez** un objet dans la mémoire en utilisant un constructeur
2. Utilisation d'objets
   - Vous pouvez appeler des méthodes
3. Destruction d'objets
   - L'objet est reconverti en mémoire
   - La mémoire est désallouée

Objets et portée

- La durée de vie d'une valeur locale est liée à l'étendue dans laquelle elle est déclarée
  - Durée de vie courte (généralement)
  - Création et la destruction déterministe
- La durée de vie d'un objet dynamique n'est pas lié à son champ d'application
  - Une durée de vie plus longue
  - Une destruction non-déterministe

Garbage Collection

- Vous ne pouvez pas détruire explicitement les objets
  - C# ne prend pas de contraire de **new** (Tels que **delete**)
  - C'est parce que d'une fonction de suppression explicite est la principale source d'erreurs dans d'autres langues
- Le Garbage Collector détruit les objets pour vous
  - Il trouve des objets non référencés et les détruit pour vous
  - Il invoque la méthode finalize sur eux pour libérer la mémoire heap (tas)
  - Il le fait généralement quand l'espace mémoire devient faible

Gestion des ressources

- Nettoyage des objet
- Rédaction des destructeurs
- Avertissements à propos du destructeur
- L'interface IDisposable et méthode Dispose
- L'instruction using en C #

Objet nettoyage

- Les actions finales des différents objets seront différentes
  - Elles ne peuvent pas être déterminés par le garbage collector.
  - Objets en. NET Framework ont une méthode **Finalise**.
  - S'il est présent, la garbage collector va appeler le destructeur avant la récupération de la mémoire (pour une autre utilisation future).
  - En C#, Implémentez un destructeur pour écrire du code de nettoyage. Vous ne pouvez pas appeler ou surcharger la méthode **Object.Finalize**.

Rédaction destructeurs

- Un destructeur est le mécanisme de nettoyage
  - Il a sa propre syntaxe:
    - Pas de modificateur d'accès
    - Aucun type de retour, pas même **void**
    - Même nom que le nom de la classe avec « ~ »
    - Pas de paramètres

```
SourceFile de classe
{
    ~ SourceFile () {...}
}
```

technofutur TIC

WAVENET
Intelligent IT Solutions

Avertissements propos de Destructeur

- L'ordre et la date de destruction n'est pas défini
  - Pas nécessairement l'inverse de la construction
- Les destructeurs sont garantis d'être appelé
  - Vous ne pouvez pas anticiper le moment où ce sera fait
- Évitez si possible les destructeurs
  - Coûteux en performance
  - Complexe
  - Retarde la libération des ressources mémoire

technofutur TIC

WAVENET
Intelligent IT Solutions

IDisposable interface et méthode Dispose

- Pour récupérer une ressource:
  - Hériter de **IDisposable** Interface et mettre en œuvre **Dispose** méthode qui libère des ressources
  - Appel **GC.SuppressFinalize** méthode
  - Veiller à ce que les appels vers **Dispose** sont fait !
  - Assurez-vous que vous n'essayez pas d'utiliser une ressource recyclée

technofuturTIC

WAVENET
Intelligent IT Solutions

L'instruction using en C

- Syntaxe

```
using (Resource r1 = new Resource( ))
{
      r1.Method( );
}
```

- Dispose est automatiquement appelée à la fin de l'aide de bloc

technofuturTIC

WAVENET
Intelligent IT Solutions