Module 4

Gestion des Exceptions

Sommaire

- Gestion des Exceptions
- Lever des Exceptions

Leçon 1: Gestion des Exceptions

- Qu’est-ce qu’une Exception?
- Utiliser un Bloc Try/Catch
- Utiliser les Propriétés des Exceptions
- Utiliser un Bloc Finally
- Utiliser les mots clés Checked et Unchecked

Qu’est-ce qu’une Exception?

Une exception est une indication d’une erreur ou une condition exceptionnelle, comme par exemple essayer d’ouvrir un fichier qui n’existe pas

Lorsqu’une méthode lève une exception, le code appelant doit être prêt à détecter et gérer l’exception

Si le code appelant ne peut pas gérer l’exception, l’exception est automatiquement propagée au code qui a appelé le code appelant

L’exception est propagée jusqu'à ce qu’une section de code gère l’exception

Si aucun code ne gère l’exception, le runtime signale qu'une exception est non gérée et l’application se plante

Utiliser un Bloc Try/Catch

```
try
{
    // Try block.
}
catch (DivideByZeroException ex)
{
    // Catch block, can access DivideByZeroException
    // exception in ex.
}
catch (Exception ex)
{
    // Catch block, can access exception in ex.
}
```

Logique qui peut générer une exception

Logique à exécuter en cas d'exception DivideByZeroException

Logique à exécuter pour toutes les autres exceptions

Vous pouvez aussi imbriquer les blocs try/catch

Les Types d'exceptions peuvent mettre en place une hiérarchie des exceptions

Utiliser les Propriétés des Exceptions

Propriétés communes:

- **Message**
- **Source**
- **StackTrace**
- **TargetSite**
- **InnerException**
- **HelpLink**
- **Data**

```
try
{
    // Try block.
}
catch (DivideByZeroException ex)
{
    Console.WriteLine(ex.Message);
}
```

Obtenir le message associé à l'exception

Utiliser un Bloc Finally

Il vous permet de libérer des ressources et de spécifier le code qui s'exécute toujours, si une exception se produit ou pas

```
try
{
    OpenFile("MyFile"); // Open a file
    WriteToFile(...);   // Write some data to the file
}
catch (IOException ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    CloseFile("MyFile"); // Close the file
}
```

Logique s'exécutera toujours

Utiliser les mots clés Checked et Unchecked

Les applications C# s'exécutent avec une vérification de dépassement de capacité des entiers désactivée par défaut. Vous pouvez modifier ce paramètre au niveau du projet

Vous pouvez contrôler le débordement pour un code spécifique en utilisant les mots clés **checked** et **unchecked**

```
checked
{
    int x = ...;
    int y = ...;
    int z = ...;
    ...
}
```

Blocs **checked** et **unchecked**

```
unchecked
{
    int x = ...;
    int y = ...;
    int z = ...;
    ...
}
```

```
...
int z = checked(x* y);
...
```

Opérateurs **checked** et **unchecked**

```
...
int z = unchecked(x* y);
...
```

Leçon 2: Lever des Exceptions

- Créer un objet Exception Object
- Lancer une Exception
- Bonnes Pratiques pour Gérer et Lever des Exceptions

Créer un objet Exception Object

System.Exception

System.SystemException

System.FormatException

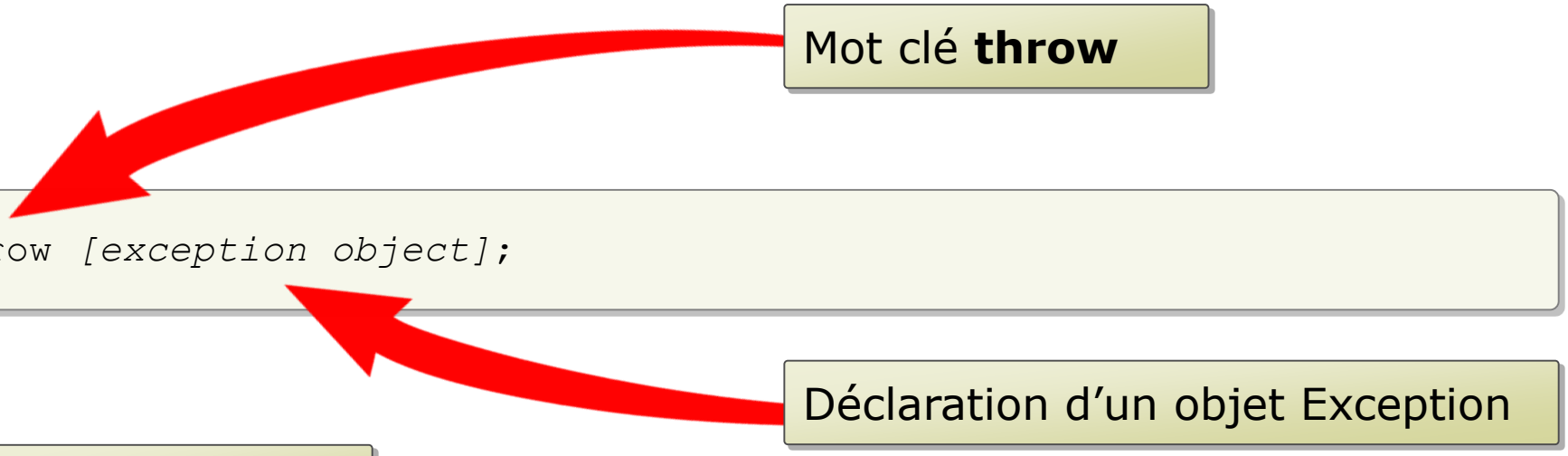
System.ArgumentException

System.NotSupportedException

Et beaucoup plus encore

```
catch (Exception e)
{
    FormatException ex =
        new FormatException("Argument has the wrong format", e);
}
```

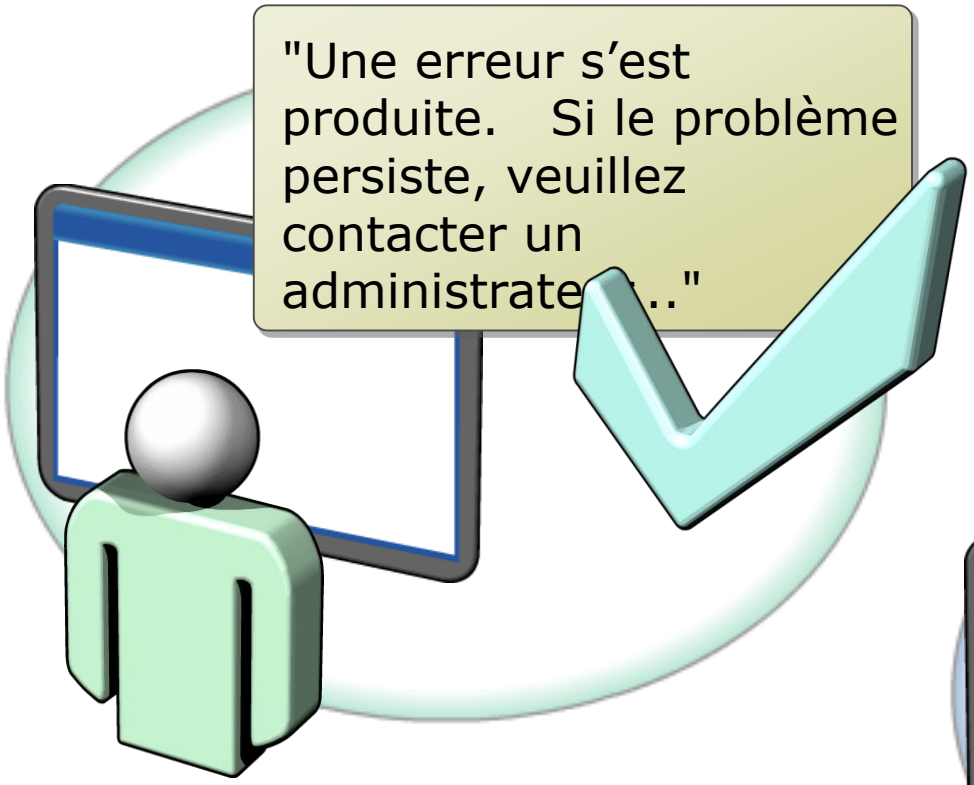
Lancer une Exception

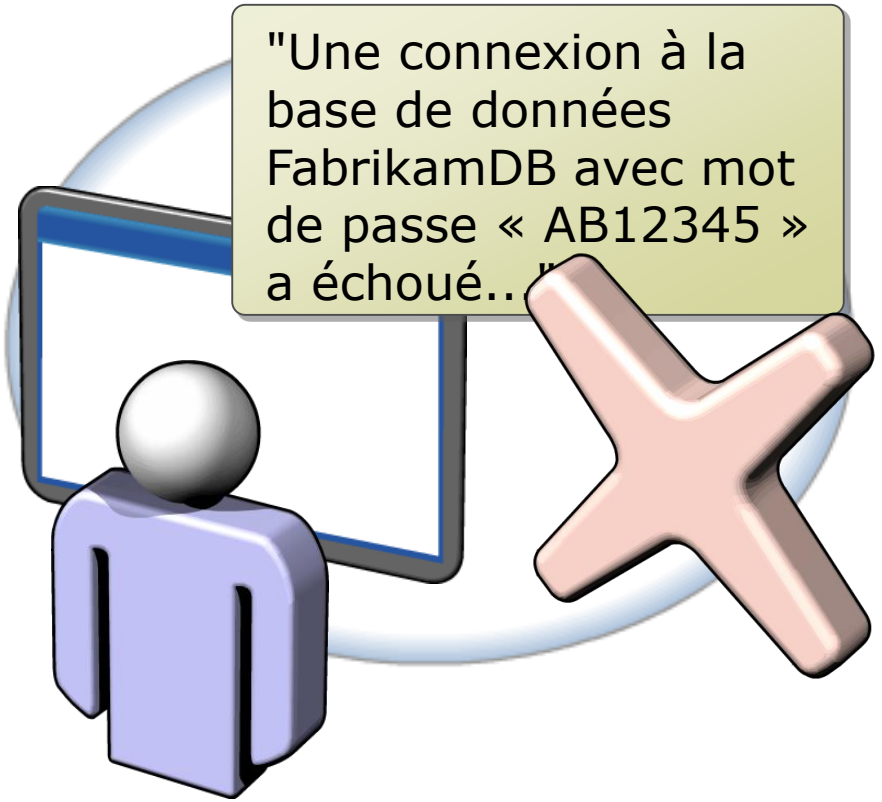


Exemple

```
public int GetIntegerRoot(int operand)
{
    double root = Math.Sqrt(operand);
    if (root != (int)root)
    {
        throw new ArgumentException("No integer root found.");
    }
    return (int)root;
}
```

Bonnes Pratiques pour Gérer et Lever des Exceptions





Ateleier Pratique

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