Module 4:
Instructions et exceptions

Vue d'ensemble

- Présentation des instructions
- Utilisation des instructions de sélection
- Utilisation des instructions d'itération
- Utilisation des instructions de saut
- Gestion des exceptions de base
- Déclencher des Exceptions

Présentation

- Les blocs d'instructions
- Types d’états

Les blocs d'instructions

- Utiliser des accolades comme délimiteurs de bloc

```
{
   / / Code
}
```

- Un bloc et son bloc parent ne peuvent pas avoir une variable du même nom

```
{
   int i;
   ...
   {
      int i;
      ...
   }
}
```

- Les blocs de mêmes parents peuvent avoir des variables avec le même nom

```
{
   int i;
   ...
}
...
{
   int i;
   ...
}
```

Types d'états

Instructions de sélection

Les instructions if et switch

Instructions d'itération

Les instructions while, do, for et foreach

Les instruction de saut

Les instructions goto, break et continue

Utilisation des instructions de sélection

- L'instruction if
- Les if en cascade
- Le switch
- Quiz: Découvrir les Bugs

L'instruction "if"

- Syntaxe:

```
if ( Boolean-expression )
  first-embedded-statement
else
  second-embedded-statement
```

- Aucune conversion implicite de int à bool

```
int x;
...
if (x) ...          // Must be if (x != 0) in C#
if (x = 0) ... // Must be if (x == 0) in C#
```

Les “if” en cascade

```
enum Suit { Clubs, Hearts, Diamonds, Spades }
Suit trumps = Suit.Hearts;
if (trumps == Suit.Clubs)
    color = "Black";
else if (trumps == Suit.Hearts)
    color = "Red";
else if (trumps == Suit.Diamonds)
    color = "Red";
else
    color = "Black";
```

Le “switch”

- Utilisez le switch pour plusieurs blocs de cas
- Utilisez l'instruction break pour s'assurer de quitter le switch

```
switch (trumps) {
case Suit.Clubs :
case Suit.Spades :
    color = "Black"; break;
case Suit.Hearts :
case Suit.Diamonds :
    color = "Red"; break;
default:
    color = "ERROR"; break;
}
```

Quiz: Découvrir les Bugs

```
if number % 2 == 0    ...
```

1

```
if (percent < 0) || (percent > 100) ...
```

2

```
if (minute == 60);
    minute = 0;
```

3

```
switch (trumps) {
case Suit.Clubs, Suit.Spades :
    color = "Black";
case Suit.Hearts, Suit.Diamonds :
    color = "Red";
defualt :
    ...
}
```

4

Utilisation des instructions d'itération

- L’instruction while
- L’instruction do
- L’instruction for
- L'instruction foreach
- Quiz: Découvrir les Bugs

L'instruction «while»

- Exécute des instructions si la valeur booléenne est vrai
- Evalue l'expression booléenne au début de la boucle
- Exécute des instructions intégrées tant que la valeur booléenne est vraie

```
int i = 0;
while (i <10) {
    Console.WriteLine (i);
    i++;
}
```

```
0 1 2 3 4 5 6 7 8 9
```

L'instruction « do »

- Exécute des instructions une 1ère fois quoi qu'il arrive.
- Evalue l'expression booléenne à la fin de la boucle.
- Exécute des instructions intégrées tant que la valeur booléenne est vraie.

```
int i = 0;
Do {
    Console.WriteLine (i);
    i++;
} While (i <10);
```

```
0 1 2 3 4 5 6 7 8 9
```

L'instruction « for »

1. **Opération d'initialisation :** création d'une variable "compteur"
2. **Opération avant chaque iteration :** test de cette variable pour determiner si une nouvelle iteration doit être réalisée
3. **Opération après chaque iteration :** incrémente le compteur

```
for (int i = 0; i <10; i++) {
    Console.WriteLine (i);
}
```

```
0 1 2 3 4 5 6 7 8 9
```

L'instruction "foreach"

- Choisissez le type et le nom de la variable d'iteration
- Exécuter des instructions pour chaque élément de la classe de collection

```
ArrayList numbers = new ArrayList( );
for (int i = 0; i < 10; i++ ) {
    numbers.Add(i);
}

foreach (int number in numbers) {
    Console.WriteLine(number);
}
```

```
0 1 2 3 4 5 6 7 8 9
```

Quiz: Découvrir les Bugs

```
for (int i = 0, i < 10, i++)
    Console.WriteLine(i);
```

1

```
int i = 0;
while (i < 10)
    Console.WriteLine(i);
```

2

```
for (int i = 0; i >= 10; i++)
    Console.WriteLine(i);
```

3

```
do
    ...
    string line = Console.ReadLine( );
    guess = int.Parse(line);
while (guess != answer);
```

4

Utilisation des instructions de saut

- L'instruction goto
- Les instructions break et continue

L'instruction goto

- Flux de contrôle transféré à une instruction étiquetée
- Peut facilement entraîner du code "spaghetti“ => **Déconseillé !**

```
if (number % 2 == 0) goto Even;
Console.WriteLine("odd");
goto End;
Even:
Console.WriteLine("even");
End:;
```

Les instructions break et continue

- L'instruction break saute d'une iteration
- L'instruction continue saute à l'itération suivante

```
int i = 0;
while (true) {
    Console.WriteLine(i);
    i++;
    if (i < 10)
        continue;
    else
        break;
}
```

Exercice: Utilisation d'instructions

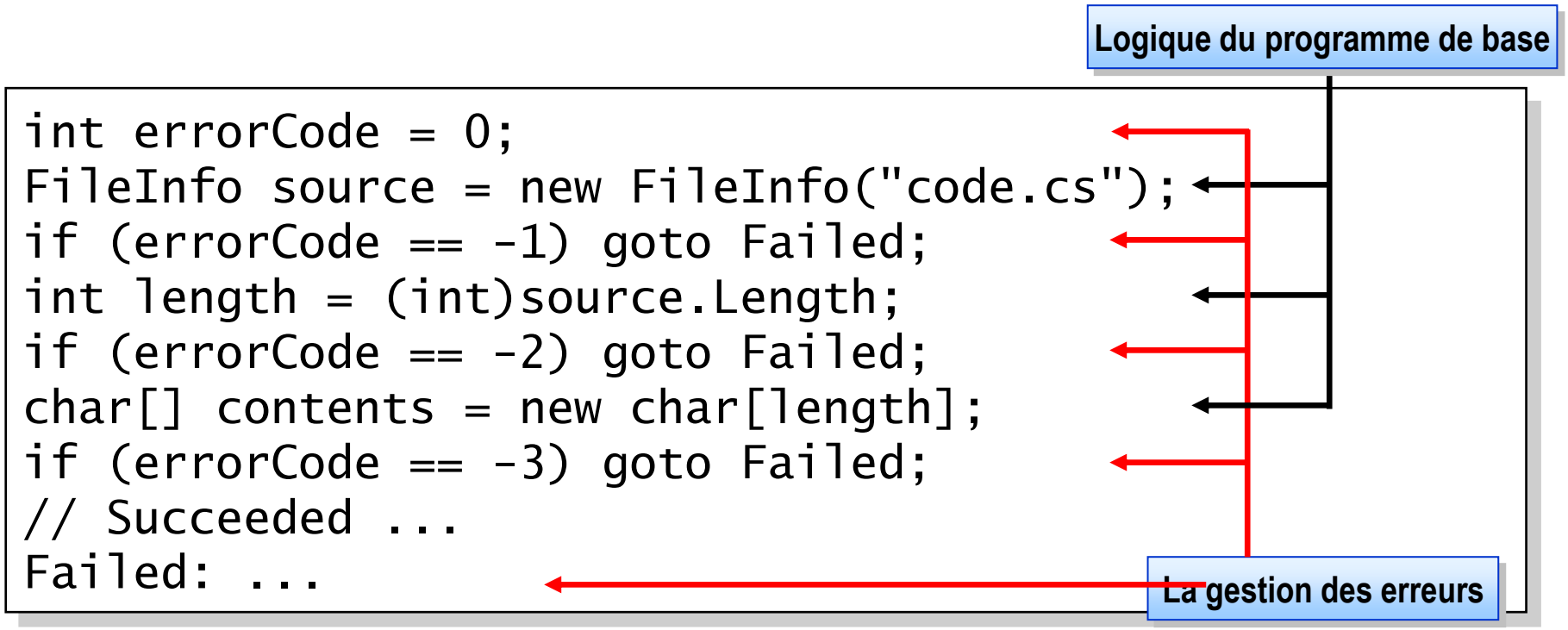
- Créer une application qui demande un nombre entre 1 et 365 à l'utilisateur.
- L'application devra convertir ce jour de l'année en une association Jour/Mois
- Exemple :
- Le nombre 93 donnera « 3 Avril »

Gestion des exceptions de base

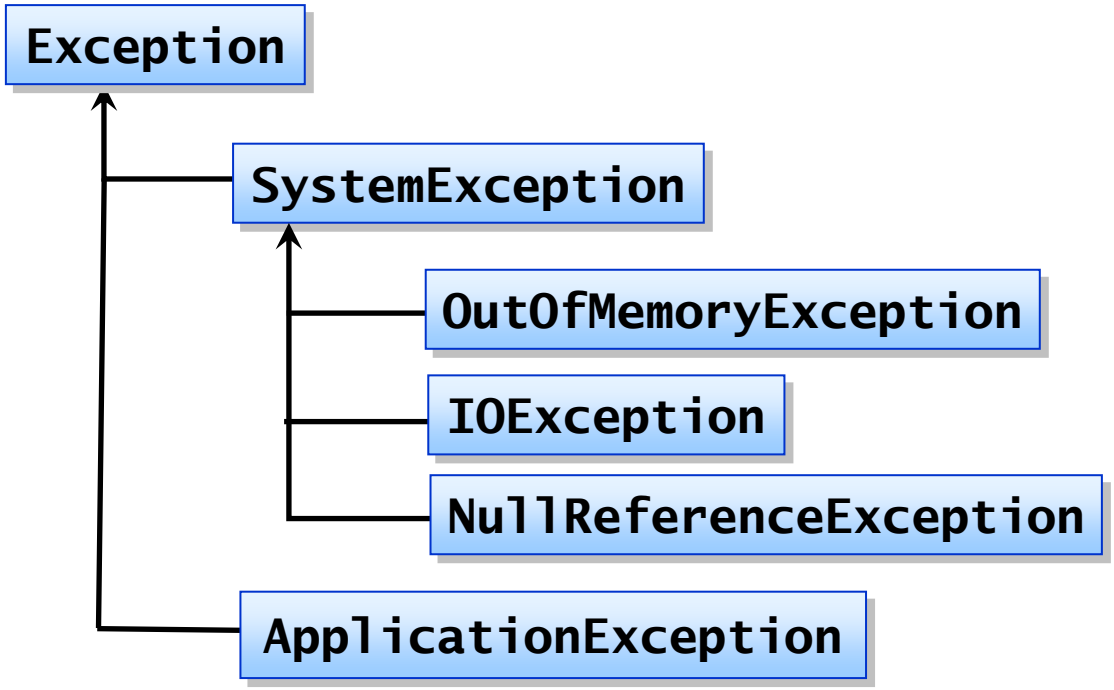
- Pourquoi utiliser des exceptions?
- Objets d'exception
- Utilisation de blocs try et catch
- Plusieurs blocs catch

Pourquoi utiliser des exceptions?

- La gestion des erreurs de procédure traditionnelle est loured.



Objets d'exception



Utilisation de blocs try et catch

- Solution orientée objet pour la gestion des erreurs
  - Mettez le code normal dans un bloc **try**
  - Gérer les exceptions dans un bloc **catch**

```
try {
        Console.WriteLine("Entrez un nombre");
        int i = int.Parse(Console.ReadLine ());
}
catch (OverflowException caught)
{
        Console.WriteLine (caught);
}
```

logique du programme

La gestion des erreurs

Plusieurs blocs catch

- Chaque bloc catch intercepte une classe d'exception
- Un bloc try peut avoir un bloc catch general
- Un bloc try n'est pas autorisé à prendre une classe qui est dérivée d'une classe pris dans un bloc catch.

```
try
{
        Console.WriteLine("Enter first number");
        int i = int.Parse(Console.ReadLine());
        Console.WriteLine("Enter second number");
        int j = int.Parse(Console.ReadLine());
        int k = i / j;
}
catch (OverflowException caught) {…}
catch (DivideByZeroException caught) {…}
```

Déclencher des Exceptions

- L'instruction throw
- La clause final
- Vérification de dépassement arithmétique
- Lignes directrices pour la gestion des exceptions

L'instruction throw

- Throw : une exception appropriée
- L'exception donne un message significatif.

```
Throw expression ;
```

```
if (minutes <1 | | minute> = 60) {
  throw new InvalidTimeException (minute +
                                  »N'est pas une minute valide");
  / /! Pas atteint!
}
```

La clause finale

- Toutes les déclarations contenues dans un bloc finally sont toujours exécutées

```
Monitor.Enter (x);
try {
    ...
}
finally {
    Monitor.Exit (x);
}
```

Tous les blocs de capture sont facultatifs

Vérification de dépassement arithmétique

- Par défaut, dépassement arithmétique n'est pas cochée
  - Une déclaration “checked” vérifie le dépassement de la limite

```
checked {
    int number = int.MaxValue;
    Console.WriteLine(++number);
}
```

OverflowException

L’exception est levée.
WriteLine est *pas* exécutée.

```
unchecked {
    int number = int.MaxValue;
    Console.WriteLine(++number);
}
```

BorneMax + 1 est négatif?

-2147483648

Lignes directrices pour la gestion des exceptions

- Throw
  - Éviter des exceptions pour les cas normaux ou prévus
  - Ne jamais créer et lancer des objets de la classe **Exception**
  - Inclure une description dans un objet **Exception**
  - Jeter des objets de la classe la plus spécifique possible
- Catch
  - Placer chaque blocs catch du plus particulier au général
  - Ne laissez pas des exceptions se lancer dans **Main**

Atelier 4.2: Utilisation d'exceptions

- Améliorer l’exercice 4.1.
- Demander l’année à l’utilisateur. S’il s’agit d’une année bissextile, gérer ce cas.
- Gérer toutes les exceptions possibles et afficher des messages clairs lorsqu'elle se produisent.