**1. Les tableaux**

**1.1 Les tableaux à une dimension**

Un tableau C# est un objet qui rassemble sous un **même nom** des données de **même type**.

**Type[ ]** *nomTtableau***[ ]=new Type[** *n* **] {** *ListeDeValeurs, ... , ...***} ;**

* *nomTableau* est choisi par le programmeur
* n = nombre de cellules du tableau.
* *nomTableau[i]* désigne la donnée n° i avec i ϵ à l'intervalle [0,n-1]

**int[ ]** *tabEntiers* **=new int[ ] {0,10,20,30};**

*ou encore*

**int[]** *tabEntiers* **= {0,10,20,30};**

* Toute référence à *Tableau[i]* où *i* n'appartient pas à l'intervalle [0,n-1]provoque une erreur de rang
* On peut déclarer ET initialiser le tableau dans le même ordre
* la propriété ***Length*** renvoie le nombre d'éléments du tableau
* la méthode **Array.Sort**(*nomTableau*) permet de trier les éléments

**1.2 Les tableaux à deux dimensions**

**Type[ , ]** *nomTtableau* **=new Type[** *n, m* **] {** *ListeDeValeurs, ... , ...***} ;**

* n = nombre de lignes du tableau
* m = nombre de colonnes du tableau
* *nomTableau[n,m]* désigne l'élément la ligne *n* et de la colonne *m*
* Le tableau à 2 dimensions peut être initialisé en même temps qu'il est déclaré
* le nombre total d'éléments avec la propriété ***Length***.

**double[ , ]** *tabRréels* **=**

**{ {0.5, 1.7}, {8.4, -6}};**

* Le nombre d'éléments dans chacune des dimensions peut être obtenue par la méthode ***GetLength***
* Le nombre de dimensions est obtenu avec la propriété ***Rank***

**1.3 Les tableaux de tableaux**

**Type[ ] [ ]** *nomTtableau* **=new Type[** *n* **] [ ];**

* n = nombre de lignes du tableau
* Impossible d'initialiser les références tableau lors de sa déclaration
* Elles ont pour valeur la référence null

**string[ ] [ ] noms = new string[3] [ ];**

**1.4 Un exemple de gestion de tableau**

// Tableau à 1 dimension, initialisé

int[] tabEntiers = new int[ ] { 0, 10, 20, 30 };

for (int i = 0; i < tabEntiers.Length; i++)

{

Console.Out.WriteLine("tabEntiers[{0}]={1}", i, tabEntiers[i]);

}

// Tableau à 2 dimensions, initialisé

double[ , ] tabReels = new double[ , ] { { 0.5, 1.7 }, { 8.4, -6 } };

for (int i = 0; i < tabReels.GetLength(0); i++)

{

for (int j = 0; j < tabReels.GetLength(1); j++)

{

Console.Out.WriteLine("réels[{0},{1}]={2}", i, j, tabReels[i, j]);

}

// Tableau de tableaux de chaines

string[ ] tabPrenom = new string[3]

{ "François", "Romain", "Julien" };

string[ ] tabNom = new string[3]

{ "Cheng", "Roland", "Gracq" };

string[ ][ ] tabPersonne =

{

tabPrenom, tabNom

};

Console.WriteLine();

for (int i = 0 ; i < 3; i++)

{

Console.WriteLine("Un auteur : " + tabPersonne[0][i] + " "

+ tabPersonne[1][i]); }

**2. Les énumérations**

**2.1 Le type enum**

Une énumération est un type de données associées à des valeurs constantes.

*Exemples : Les jours de la semaines, les mois de l'année, les mentions possibles obtenues au baccalauréat*

On peut définir une énumération pour ces constantes :

enum Jours { Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche};

enum Mois { Janvier, février, .... Novembre, Décembre};

enum Mentions { Passable, AssezBien, Bien, TrèsBien, Excellent };

**Mentions** *maMention* **= Mentions.Passable;**

Affectation à une variable d'une des constantes de l'énumération

**if (***maMention* **== Mentions.Passable)**

**{Console.WriteLine("Peut mieux faire"); }**

Comparaison d'une variable aux valeurs possibles de l'énumération

La méthode**GetValues**permet d'obtenir toutes les valeurs de l'énumération

**foreach (Mentions** *m* **in Enum.GetValues(maMention.GetType()))**

**{ Console.WriteLine(***m***); }**

m est un objet de type "Mentions"

**foreach (int** *m* **in Enum.GetValues(typeof(Mentions))**

**{ Console.WriteLine(***m***); }**

la méthode **typeof** renvoie le rang de la constante dans l'énumération

**2.2 Exemple de type enum : les mentions aux Baccalauréat**

....

class Program

{

**enum Mentions { Passable, AssezBien, Bien, TrèsBien};** // attention se place avant le début du programme principal

static void Main(string[] args)

{

// Affectation à une variable d'une des constantes de l'énumération

Mentions maMention = Mentions.AssezBien;

// affichage valeur variable

Console.WriteLine("Affectation à la variable maMention d'une énumération : " + maMention);

// Test sur une des valeurs de l'énumération

if (**maMention == Mentions.TrèsBien**)

{ Console.WriteLine("Excellent travail"); }

else

{ Console.WriteLine("Vous pouvez mieux faire ! "); }

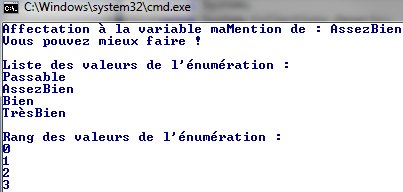
// Affichage des mentions sous forme de chaînes

Console.WriteLine("Liste des valeurs de l'énumération : ");

**foreach (Mentions m in Enum.GetValues(maMention.GetType()))**

{

Console.WriteLine(m);

 }

// Affichage des rangs des mentions dans la liste

Console.WriteLine();

Console.WriteLine("Rang des valeurs de l'énumération : ");

foreach (**int** m in Enum.GetValues(**typeof**(Mentions)))

{

Console.WriteLine(m);

} } } }

**Trace d'exécution**

**3. Exercices d'application**

**3.1 Tableau de nom d'employés**

Vous devez produire un programme qui permette de gérer dans un tableau les noms des employés de l'entreprise BOTAB

Les données à ranger dans le tableau sont les valeurs suivants (prévoir 5 lignes pour le tableau)

|  |  |
| --- | --- |
| 10 | Romain Poivre |
| 20 | Pauline Safran |
| 30 | Coline Cannelle |
| 40 | Joël Coriandre |

Le tableau étant créé et valorisé, vous affichez son contenu : N° et nom du salarié

Vous devez insérer un nouvel élément entre Pauline Safran et Coline Cannelle : 25 Dorian Cumin

**A faire :** Ecrire le programme C# permettant de réaliser ces objectifs de gestion de tableau

**3.2 Enumération de types d'animaux**

L' association SHAT héberge des animaux abandonnés : chats, chiens, chevaux, lapins et oiseaux.

SHAT souhaite gérer les types d'animaux qui lui sont confiés. Un animal sera associé à un type.

**A faire :** Ecrire le programme C# permettant de gérer le type des animaux à l'aide d'une liste.

On affichera le type 'chat', puis tous les types des animaux accueillis au refuge SHAT.

**3.3 Entraînement à l'examen : afficher la liste alphabétique des noms des employés**

**A faire :**

Quel problème va-t-on rencontrer pour trier les salariés sur leur nom ?

Comment peut-on résoudre ce problème ?

Produire à l'écrit l'algorithme qui crée un tableau unidimensionnel contenant les noms des salariés

Ajouter au programme précédemment réalisé, les méthodes C# permettant :

- d'obtenir le tableau unidimensionnel

- de trier les éléments de ce tableau

- d'afficher les éléments triés

***Travail à finir pour lundi 09/09/2013* : constituer le dossier correspond au travail à faire**

**3. Gestion d'une exception**

**3.1 Les ordres du langage C# : try , catch, finally[[1]](#footnote-1)**

De nombreuses fonctions C# peuvent générer des exceptions, c'est à dire des erreurs détectées par le CLR[[2]](#footnote-2).

Si une fonction génère une exception, le programme plante et affiche un message erreur que programmeur doit gérer pour éviter le "plantage" de l'application.

Exemple la méthode Convert peut générer une exception

**try**

{

*code susceptible de générer une exception*

}

**catch** (Exception e)

{

*traiter l'exception e*

}

*instruction suivante*

La gestion d'une exception se fait selon le formalisme ci-contre :

Si la fonction ne génère pas d'exception = pas d'erreur détectée :

* le programme continue à l'*instruction suivante*

Si erreur détectée :

* les instructions du *catch* sont exécutées
* puis le programme continue à l'*instruction suivante*

**try**

{

*code susceptible de générer les exceptions*

}

**catch** ( IndexOutOfRangeException e1)

{

*traiter l'exception e1*

}

**catch** ( FormatException e2)

{

*traiter l'exception e2*

}

instruction suivante

Il existe plusieurs types d'exceptions

On peut être plus précis en utilisant des types tels que :

* *IndexOutOfRangeException,*
* *FormatException,*
* *SystemException,*.

En écrivant *catch (Exception e)*, on indique qu'on veut gérer toutes les types d'exceptions.

Si le code de la clause *try* est susceptible de générer plusieurs types d'exceptions, on peut associer l'exception avec plusieurs clauses *catch*

**try**

{

*code susceptible de générer une exception*

}

**catch** (Exception e)

{

*traiter l'exception e*

}

**finally**

{

*code exécuté après try ou catch*

}

On peut ajouter aux clauses *try/catch*, une clause **finally**

Qu'il y ait exception ou pas, le code de la clause *finally* sera toujours exécuté.

La classe *Exception* a une propriété **Message** : affiche un message détaillant l'erreur qui s'est produite

La classe *Exception* a une méthode **ToString :** renvoie une chaîne de caractères indiquant le type de l'exception ainsi que la valeur de la propriété *Message*.

**3.2 Exemple : traitement de l'âge avec gestion d'exception sur la méthode Convert**

using System; using System.Collections.Generic; using System.Text;

namespace Exception\_Age

{

class Program

{

static void Main(string[] args)

{

// initialisation des variables

string prenom;

int age = 0;

bool ageOK = false;

// Saisie du prenom

Console.Write("Votre prénom : ");

prenom = Console.ReadLine();

// -------------------------------------------------------------------

// Tant Que la saisie de l'âge génère une erreur

while (!ageOK)

{

// Saisie de l'âge

Console.Write("Votre âge : ");

// Vérification de la valeur de la variable âge saisie et gestion d'exception

**try**

{

// Si conversion impossible, Convert provoque une exception ==> l'opération est placée dans un try-catch.

age = **Convert**.ToInt16(Console.ReadLine());

ageOK = age >= 17;

if (!ageOK)

{

Console.WriteLine("Votre âge semble incorrect, recommencez ...");

}

}

**catch** (Exception)

{

Console.WriteLine("Vous devez saisir une valeur numérique entière");

}

// Fin de la gestion d'exception

}

// Fin du Tant Que

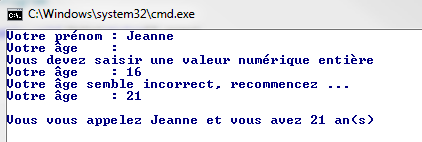
// ----------------------------------------------------------------------

// affichage final

Console.WriteLine();

Console.WriteLine("Vous vous appelez {0} et vous avez {1} an(s)",prenom,age);

Console.ReadKey();

 }

}

}

**Trace d'exécution**

1. Voir le glossaire C# pour le format des ordres try / catch [↑](#footnote-ref-1)
2. CLR : **Common Language Runtime**  = définit l'environnement d'exécution des programmes sur la plateforme .NET

   Le CLR fait tourner une sorte de bytecode nommé Common Intermediate Language (CIL). Le compilateur à la volée transforme le code CIL en code natif spécifique au système d'exploitation. [↑](#footnote-ref-2)