Les interfaces

Les méthodes d’une interface sont implicitement abstraites (elles ne peuvent pas être instancié). Toutes les méthodes d’une interface doivent être définis dans la classe qui en hérite.

Considérons l’interface IStat :

interface Istat

{

double Moyenne();

double EcartType();

}

La classe NoteEleve qui hérite de Istat doit définir les méthodes Moyenne() et EcartType()

class NoteEleve : Istat

{

public List<double> notes;

public NoteEleve()

{

notes = new List<double>();

}

public double Moyenne()

{

double somme = 0.0;

for(int i = 0; i < notes.Count; i++)

{

somme += notes.ElementAt(i);

}

return somme / notes.Count;

}

public double EcartType()

{

double moyenne = Moyenne();

double somme = 0.0;

for (int i = 0; i < notes.Count; i++)

{

double delta = notes[i] - moyenne;

somme += delta \* delta;

}

return Math.Sqrt(somme / (notes.Count - 1));

}

}

Pareil pour ValeurMensuelProduit et Temperature

class ValeurMensuelProduit : Istat

{

public List<double> valeurs;

public ValeurMensuelProduit()

{

valeurs = new List<double>();

}

public double Moyenne()

{

double somme = 0.0;

for (int i = 0; i < valeurs.Count; i++)

{

somme += valeurs.ElementAt(i);

}

return somme / valeurs.Count;

}

public double EcartType()

{

double moyenne = Moyenne();

double somme = 0.0;

for (int i = 0; i < valeurs.Count; i++)

{

double delta = valeurs[i] - moyenne;

somme += delta \* delta;

}

return Math.Sqrt(somme / (valeurs.Count - 1));

}

}

class Temperature : Istat

{

public List<double> temperatures;

public Temperature()

{

temperatures = new List<double>();

}

public double Moyenne()

{

double somme = 0.0;

for (int i = 0; i < temperatures.Count; i++)

{

somme += temperatures.ElementAt(i);

}

return somme / temperatures.Count;

}

public double EcartType()

{

double moyenne = Moyenne();

double somme = 0.0;

for (int i = 0; i < temperatures.Count; i++)

{

double delta = temperatures[i] - moyenne;

somme += delta \* delta;

}

return Math.Sqrt(somme / (temperatures.Count - 1));

}

}

class Program

{

static void Note()

{

double n = 0;

string saisie = "";

NoteEleve ne = new NoteEleve();

while (saisie != "s")

{

Console.WriteLine("entrez la note d'un élève");

n = Convert.ToDouble(Console.ReadLine());

ne.notes.Add(n);

Console.WriteLine("pour arrêter la saisie, taper s");

saisie = Console.ReadLine();

}

Console.WriteLine("la moyenne = " + ne.Moyenne());

Console.WriteLine("l'écart-type = " + ne.EcartType());

}

static void VMP()

{

double v = 0;

string saisie = "";

ValeurMensuelProduit vmp = new ValeurMensuelProduit();

while (saisie != "s")

{

Console.WriteLine("entrez la valeur du produit");

v = Convert.ToDouble(Console.ReadLine());

vmp.valeurs.Add(v);

Console.WriteLine("pour arrêter la saisie, taper s");

saisie = Console.ReadLine();

}

Console.WriteLine("la moyenne = " + vmp.Moyenne());

Console.WriteLine("l'écart-type = " + vmp.EcartType());

}

static void Temp()

{

double t = 0;

string saisie = "";

Temperature temp = new Temperature();

while (saisie != "s")

{

Console.WriteLine("entrez la température");

t = Convert.ToDouble(Console.ReadLine());

temp.temperatures.Add(t);

Console.WriteLine("pour arrêter la saisie, taper s");

saisie = Console.ReadLine();

}

Console.WriteLine("la moyenne = " + temp.Moyenne()+"°");

Console.WriteLine("l'écart-type = " + temp.EcartType());

}

static void Main(string[] args)

{

Note();

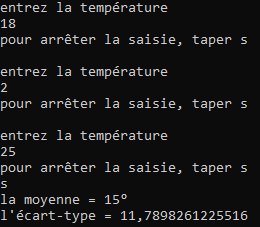
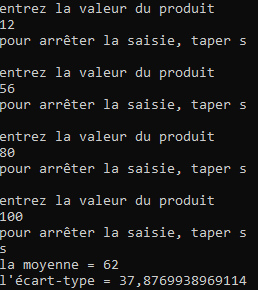
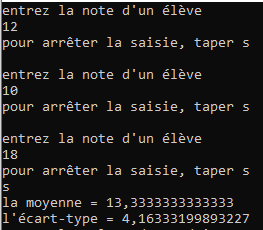
VMP();

Temp();

}

}

# Screenshot



# Interfaces de visual studio

Dans visual studio, les interfaces commencent toutes par un I majuscule et ont un pictogramme de la forme d’une clé bleue devant leur nom. Par exemple, on a : IAppDomainSetup, IAsyncResult, ICloneable, IComparer, IConvertible, ICustomFormatter, IDictionary, etc…

class CasPerso : IList

{

private object[] \_contents = new object[8];

private int \_count;

public CasPerso()

{

\_count = 0;

}

// IList Members

public int Add(object value)

{

if (\_count < \_contents.Length)

{

\_contents[\_count] = value;

\_count++;

return (\_count - 1);

}

else

{

return -1;

}

}

public void Clear()

{

\_count = 0;

}

public bool Contains(object value)

{

bool inList = false;

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

{

if (\_contents[i] == value)

{

inList = true;

break;

}

}

return inList;

}

public int IndexOf(object value)

{

int itemIndex = -1;

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

{

if (\_contents[i] == value)

{

itemIndex = i;

break;

}

}

return itemIndex;

}

public void Insert(int index, object value)

{

if ((\_count + 1 <= \_contents.Length) && (index < Count) && (index >= 0))

{

\_count++;

for (int i = Count - 1; i > index; i--)

{

\_contents[i] = \_contents[i - 1];

}

\_contents[index] = value;

}

}

public bool IsFixedSize

{

get

{

return true;

}

}

public bool IsReadOnly

{

get

{

return false;

}

}

public void Remove(object value)

{

RemoveAt(IndexOf(value));

}

public void RemoveAt(int index)

{

if ((index >= 0) && (index < Count))

{

for (int i = index; i < Count - 1; i++)

{

\_contents[i] = \_contents[i + 1];

}

\_count--;

}

}

public object this[int index]

{

get

{

return \_contents[index];

}

set

{

\_contents[index] = value;

}

}

// ICollection Members

public void CopyTo(Array array, int index)

{

int j = index;

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

{

array.SetValue(\_contents[i], j);

j++;

}

}

public int Count

{

get

{

return \_count;

}

}

public bool IsSynchronized

{

get

{

return false;

}

}

// Return the current instance since the underlying store is not

// publicly available.

public object SyncRoot

{

get

{

return this;

}

}

// IEnumerable Members

public IEnumerator GetEnumerator()

{

// Refer to the IEnumerator documentation for an example of

// implementing an enumerator.

throw new Exception("The method or operation is not implemented.");

}

public void PrintContents()

{

Console.WriteLine("La liste a une capacité de {0} et est composé de {1} éléments.", \_contents.Length, \_count);

Console.Write("La liste contient :");

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

{

Console.Write(" {0}", \_contents[i]);

}

Console.WriteLine();

}

}

## Programme de test

static void Main(string[] args)

{

CasPerso test = new CasPerso();

// Populate the List

Console.WriteLine("Ajout d'élément à la liste");

test.Add("one");

test.Add("two");

test.Add("three");

test.Add("four");

test.Add("five");

test.Add("six");

test.Add("seven");

test.Add("eight");

test.PrintContents();

Console.WriteLine();

// Remove elements from the list

Console.WriteLine("Suppression d'éléments de la liste");

test.Remove("six");

test.Remove("eight");

test.PrintContents();

Console.WriteLine();

// Add an element to the end of the list

Console.WriteLine("Ajout d'un élément à la fin de la liste");

test.Add("nine");

test.PrintContents();

Console.WriteLine();

// Insert an element into the middle of the list

Console.WriteLine("Ajout d'un élément au milieu de la liste");

test.Insert(4, "number");

test.PrintContents();

Console.WriteLine();

// Check for specific elements in the list

Console.WriteLine("Recherche d'un élément spécifique dans la liste");

Console.WriteLine("La liste contient \"three\": {0}", test.Contains("three"));

Console.WriteLine("La liste contient \"ten\": {0}", test.Contains("ten"));

Console.Read();

}