ECAM
Brussels Engineering School

mercredi 20 septembre 2017
LUR, XIN

E303D Laboratoire de concepts informatiques

Labo 1 : Découverte et prise en main du C#

Le but de cette première séance de laboratoire est de découvrir le langage C# et de le prendre en main. On va y revoir les concepts de l'orienté objet vus en deuxième année, mais appliqués en C# et on va construire une première interface graphique.

Objectifs

- ☐ Maitrise des **bases du langage C#** : savoir refaire ce qu'on a vu en programmation orientée objet avec Python (chapitres 8 et 9 du livre de référence).
- ☐ Prise en main d'un **IDE** entre Microsoft Visual Studio Community et Xamarin Studio : création d'un nouveau projet, compilation et exécution.
- ☐ Réalisation d'une première **interface graphique** : placement de composants et liaison d'un gestionnaire d'évènements.

Hello World

Pour les premiers exercices, on va réaliser des programmes qui fonctionneront uniquement en console, c'est-à-dire sans interface graphique. Sous Microsoft Visual Studio Community, créez un nouveau projet C# de type *« Console Application »* et sous Xamarin Studio, créez une nouvelle solution C# de type *« Projet console »*. Un fichier `Program.cs` est automatiquement créé pour vous. Ce dernier contient une classe avec une méthode de classe (`static`) appelée `Main` qui est en fait le point d'entrée du programme, c'est-à-dire que c'est son corps qui sera exécuté lorsque vous exécutez le programme. Ajoutez-y l'instruction suivante, et exécutez le programme pour voir s'afficher la phrase `Hello World!` dans la console à l'écran :

```
1 Console.WriteLine("Hello World!");
```

On repère immédiatement quelques éléments de syntaxe du langage et également des conventions :

— on doit délimiter les blocs de code avec des accolades ;
— les instructions se terminent par un point-virgule ;
— les noms de classe et de méthodes commencent par une majuscule ;
— les noms de variable commencent par une minuscule.

Constructions de base du langage

Vous allez maintenant devoir chercher comment on construit les différents éléments suivants, que vous connaissez déjà en Python, avec le langage C#. Le but final sera d'être capable de refaire le programme qui calcule les racines d'un trinôme du second degré de la forme $ax^2 + bx + c$, repris au listing 1.

Le programme doit donc demander les trois coefficients du trinôme à l'utilisateur puis, après les avoir converti en des nombres entiers, il doit calculer et afficher les racines réelles qu'il possède. On fait, pour le moment, la supposition que l'utilisateur entre bien des nombres réels.

```python
# Programme de recherche des racines d'un trinôme
# du second degré de la forme ax^2 + bx + c
# Auteur : Sébastien Combéfis
# Version : 7 aout 2015

from math import sqrt

print("Recherche des racines de ax^2 + bx + c")
a = int(input("Coefficient a : "))
b = int(input("Coefficient b : "))
c = int(input("Coefficient c : "))

# Calcul du discriminant
delta = b**2 - 4 * a * c
print("Discriminant :", delta)

# Test des trois cas possibles et affichage des racines du trinôme
if delta < 0:
    print("Pas de racine réelle")
elif delta == 0:
    x = -b / (2 * a)
    print("Une racine réelle double :", x)
else:
    x1 = (-b - sqrt(delta)) / (2 * a)
    x2 = (-b + sqrt(delta)) / (2 * a)
    print("Deux racines réelles distinctes:", x1, "et", x2)
```

Listing 1 – Le fichier `trinomial-root.py` contient un programme de recherche des racines d'un trinôme du second degré.

Voici les constructions qu'il vous faut savoir faire avec C# :

- ☐ insérer un commentaire dans un code source ;
- ☐ variable : déclaration, initialisation et utilisation ;
- ☐ types de données : comprendre le type `double` ;
- ☐ lecture et écriture à la console avec l'objet `Console` ;
- ☐ conversion de données avec l'objet `Convert` ;
- ☐ calcul d'une racine carrée avec l'objet `Math` ;
- ☐ construction d'expressions booléennes pour définir des conditions ;
- ☐ utilisation de l'instruction `if-else if-else`.

Constructions avancées du langage

Vous devez maintenant améliorer le programme pour gérer le cas où l'utilisateur entre du texte qui n'est pas valide, c'est-à-dire qui ne représente pas un nombre. Tant qu'il n'a pas renseigné une valeur valide, le programme doit continuer à lui demander en boucle la valeur attendue. Pour cela, vous allez devoir découvrir le fonctionnement des concepts suivants :

- ☐ définition d'une fonction (déclarez la `private static`) ;
- ☐ utilisation de l'instruction `while` ;
- ☐ utilisation du mécanisme d'exception `try-catch`.

La construction `default` pourrait vous servir pour que votre code compile. Elle permet d'obtenir automatiquement la valeur par défaut associée à un type de données. Par exemple, `default (double)` vous donnera la valeur par défaut d'un `double`.

Programmation orientée objet

On va maintenant voir comment pratiquer la programmation orientée objet en C#. Vous allez devoir définir une classe `Polynomial` qui représente un polynôme $a_n x^n + ... + a_2 x^2 + a_1 x + a_0$, de degré n. On souhaiterait pouvoir exécuter le programme suivant :

```
1 Polynomial p = new Polynomial(new double[] {1, 0, -2});
2
3 Console.WriteLine(p.Degree);
4 Console.WriteLine(p);
5 Console.WriteLine(p.Evaluate(2));
```

L'exécution du programme doit afficher ce qui suit :

```
2
1x^2 + -2
2
```

La classe `Polynomial` doit donc contenir :

- un constructeur qui reçoit en paramètre un tableau de `double` (un tableau est similaire aux listes Python, sauf que sa taille ne peut pas changer une fois créé) ;
- une propriété `Degree` accessible en lecture seule qui renvoie le degré du polynôme (similaire aux accesseurs Python) ;
- une méthode `Evaluate` qui prend un nombre réel x en paramètre et renvoie le résultat de l'évaluation du polynôme en ce point ;
- une méthode `ToString` qui renvoie une représentation textuelle de l'objet (comme `__str__` en Python). Notez que la signature de la méthode doit être `public override string ToString()`.

Pour résoudre cet exercice, il vous faudra donc découvrir les concepts suivants :

- ☐ définition d'une nouvelle classe : variables d'instance, constructeur et méthode ;
- ☐ définition d'une propriété (en lecture et/ou écriture) ;
- ☐ manipulation d'un tableau.
- ☐ utilisation de l'instruction `for` ;
- ☐ calcul d'une puissance avec l'objet `Math` ;
- ☐ construction d'une chaine de caractères par concaténation ;
- ☐ principe de visibilité `private`/`public`, pour l'encapsulation.

Égalité d'objets

En C#, tout comme en Python, on peut tester si les références stockées dans deux variables réfèrent des objets qui ont la même identité ou non, et s'ils ont le même état ou non. L'identité se teste avec l'opérateur `==` et l'état avec la méthode `Equals`. L'exécution du programme suivant devrait afficher `False` puis `True` :

```
1 Polynomial p = new Polynomial(new double[] {1, 0, -2});
2 Polynomial q = new Polynomial new double[]{1, 0, -2});
3
4 Console.WriteLine(p == q);
5 Console.WriteLine(p.Equals(q));
```

Complétez la classe `Polynomial` pour que ce soit le cas. Pour résoudre cet exercice, vous devriez avoir besoin des concepts suivants :

- ☐ récupération du type d'un objet (avec la méthode `GetType`) ;
- ☐ conversion d'une référence d'un type d'objet vers un autre.

Compte à rebours

Le challenge de ce labo consiste à réaliser un compteur à rebours. Définissez une classe `CountDown` qui doit contenir :

- un constructeur qui prend un nombre de secondes en paramètre ;
- une propriété `RemainingTime` qui renvoie le temps restant du compte à rebours ;
- une méthode `Start` qui lance le compte à rebours et une méthode `Stop` qui l'arrête.

On souhaite, par exemple, pouvoir écrire les instructions suivantes :

```
1 CountDown countdown = new CountDown(600);
2
3 countdown.Start();
4 Console.WriteLine(countdown.RemainingTime);
5
6 System.Threading.Thread.Sleep(5000);
7 Console.WriteLine(countdown.RemainingTime);
```

On crée donc un nouvel objet `CountDown` qui va faire un décompte de 600 secondes. On le démarre ensuite en affichant juste après le temps restant. Ensuite, on suspend l'exécution de ces instruction pendant 5 secondes (le paramètre de la méthode `Sleep` est un nombre de millisecondes), avant d'afficher à nouveau le temps restant. L'exécution de ces instructions affiche normalement 600 suivi de 595.

Pour vous aider, vous pourriez avec besoin des éléments suivants :

- ☐ répétition d'une tâche à intervalle régulier avec l'objet `System.Timers.Timer` ;
- ☐ ajout d'une tâche à répéter avec la propriété `Elapsed` d'un `Timer` ;
- ☐ définition d'une méthode privée comme gestionnaire d'évènement, qui prend deux paramètres `Object source` et `ElapsedEventArgs e`.

S'il vous reste du temps en fin de séance, essayez de mettre au point une interface graphique composée d'une simple fenêtre qui affiche le temps qu'il reste au compte à rebours et qui permet de le démarrer et l'arrêter à l'aide de boutons.