Développement d'application Nomade (iOS & Xamarin)

COURS 04 – INTRODUCTION À C#

M2 INIS – 2015

julien.mialon@univ-orleans.fr
martinsfonseca.jose@yahoo.fr

Plan du cours

- Introduction

iOS

- Introduction à Objective-C
- Structure d'une application
- Les vues
- La navigation
- Les tables
- Persistance de données

Xamarin

- Introduction à C#
- MVVM et Xamarin.Forms
- NuGet & Xamarin Store

C#

C#

- Basé sur C++ (concept) & Java (syntaxe)
- IDE : Visual Studio / Xamarin Studio / MonoDevelop

```
using System;

namespace DemoCours
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World !");
        }
    }
}
```

C# *VS* Java

- Structure

```
struct MyStruct
{
    public int MyField;
}
```

- Surcharge d'opérateur

```
// Overloading '+' operator:
public static ComplexNumber operator+(ComplexNumber a, ComplexNumber b)
{
    return new ComplexNumber(a.real + b.real, a.imaginary + b.imaginary);
}
```

- Généricité (Templates)

```
// Declare the generic class.
public class GenericList<T>
{
    void Add(T input) { }
}
```

C# *VS* Java

- Exceptions non vérifiées
  - Pas besoin de catch toutes les exceptions imaginables
  - Pas besoin de les déclarer avec throws

- Préprocesseur

```
#warning A warning
#error An error

#if DEBUG
    // Code Debug
#else
    // Code Release
#endif
```

- string et non String (en prime le « == » fonctionne ☺)

```
string MyString;
```

C# *VS* Java

- Propriétés

```
private string _name;

public string Name
{
    get { return _name; }
    set
    {
        if (value != null)
        {
            _name = value;
        }
    }
}

public int Age { get; set; }
```

C# *VS* Java

- Delegate : signature de fonction

```
// Définition du delegate
public delegate int Compute(int x, int y);
// Méthode avec la même signature que le delegate
public static int Add(int x, int y)
{
    return x + y;
}
// Méthode utilisant le delegate
public static int Exec(int x, int y, Compute op)
{
    return op(x, y);
}
// Main
static void Main(string[] args)
{
    int r1 = Exec(2, 4, Add);
    int r2 = Exec(6, 8, (x, y) => x - y);
}
```

C# *VS* Java

- Lambda expressions
  - Définition rapide de méthode anonyme
- (param1, param…) => Instruction;
- (param1, param…) => {
- Bloc instruction
- };

```
(x, y) => x + y;

(x, y) =>
{
    x++;
    return x + y;
};
```

C# *VS* Java

- Évènements
  - Equivalent aux listeners Java => Mais plus simple ☺

```
public event EventHandler<string> OnPropertyChanged;

public void NotifyPropertyChanged(string propertyName)
{
    var handler = OnPropertyChanged;
    if (handler != null)
    {
        handler(this, propertyName);
    }
}

public void PropertyChangedCallback(object sender, string propertyName)
{
    // Do something...
}

public void Main()
{
    OnPropertyChanged += PropertyChangedCallback;
}
```

C# *VS* Java

- Indexeurs (ou opérateur [])

```
class NumberTab
{
    private int [] _tab = new int[10];

    public int this[int index]
    {
        get
        {
            return _tab[index];
        }
        set
        {
            _tab[index] = value;
        }
    }
```

```
NumberTab tab = new NumberTab();
tab[0] = 42;
```

Asynchronisme

- Méthode asynchrone => Retourne Task ou Task<T>
- Exemple : `Task<IFile> File.OpenFileAsync(string page)`

```
public async Task ReadData()
{
    IFile file = await File.OpenFileAsync("hello.txt");
    file.ReadAll();
}
```

- Async permet de déclarer une méthode comme asynchrone (le wrapping du retour en Task sera fait automatiquement)
- Await permet de forcer l'attente de la fin d'une Task et de récupérer le résultat

Asynchronisme

- Wrapping automatique en Task

```
public async Task<int> Answer()
{
    await Task.Delay(1000);
    return 42;
}
```

- Attention à bien utiliser await sur les méthode asynchrone si vous avez besoin de leur résultat :

```
public async Task ReadData()
{
    IFile file = await File.OpenFileAsync("hello.txt");
    this.Data = file.ReadAll();
}

public void DisplayContent()
{
    ReadData();
    Console.WriteLine(this.Data);
}
```

Asynchronisme

- Comment utiliser une méthode asynchrone dans une méthode qu'on ne veux pas asynchrone.

```
public async Task ReadData()
{
    IFile file = await File.OpenFileAsync("hello.txt");
    this.Data = file.ReadAll();
}

public void DisplayContent()
{
    ReadData().ContinueWith(
        (x, state) => Console.WriteLine(this.Data)
        , null);
}
```

.NET

Exécution de C#

- Un programme C# s'exécute sur un environnement .NET
- CLR => Common Language Runtime
  - Permet aussi de faire tourner F#, VB, …
- À votre disposition :
  - Toutes les API .NET pour la plateforme que vous ciblez
- Version actuelle :
  - .NET 4.5
  - C# 5.0

Questions ?