Chapitre

Organisation du cours

1 Objectifs du cours

Ce cours de *Formalisation des Connaissances et Programmation Fonctionnelle* s'attèle à vous enseigner comment:

- Formaliser des informations
- Organiser et décrire des données puis des objets complexes (typage et représentation objet)
- Utiliser les technologies XML pour structurer, décrire, stocker, transformer, utiliser ces données
- Ecrire un programme sécurisé, testable, enutilisant les paradigmes de la programmation fonctionnelle.

Ce cours utilise la norme XML pour stocker les données. UML et XMLSchema sont utilisés pour décrire et organiser les données. En se basant sur UML et XMLSchema, le lien avec la Programmation Orientée Objet (POO) est simplifié.

Notez que nous utiliserons .Net (DotNet) C# comme langage pour rendre opérationnels nos projets utilisant XML et implémenter les paradigmes de la programmation fonctionnelle. Ce cours ne constitue en aucun cas un cours avancé de programmation C#. Notez aussi que le cours contient des exemples d'utilisation des technologies XML en Java, son API présentant quelques différences importantes avec C#. Là encore, il ne s'agit pas d'un cours de Java. Vous aurez cependant des rappels réguliers de C# et Java au cours des CTD et TP.

2 Pourquoi XML ?

Pourquoi XML plutot que d'autres formats comme JSON, les expressions régulières ... ?.
Le point important, est que cet enseignement vise à vous faire prendre de bonnes habitudes de programmation, à savoir de savoir modéliser les données proprement et ensuite de savoir les utiliser. Le Web est effectivement complètement centré sur le stockage et l'échange de données. Le reste "n'est qu'interface et fonctionnalisation". C'est le cas de nombreuses autres applications. Dans ce cours, nous verrons comment interagir avec ces données, les transformer et les rendre disponibles, en utisant Java ou C#.
Maintenant, à la question de pourquoi XML et pas JSON ou autre: en fait cela dépend un peu du problème que l'on traite. Je vous renvoie pour cela à ce lien fort clair et fort bien documenté. Cependant, il apparaît qu'XML est adapté à l'objectif principal de ce cours (formalisation, structuration des données, mais également navigation, usage ...). XML dispose d'atouts moins marqués ou absents en JSON :

- XML permet (en XML Schema) de décrire des types complexes, incluant des notions d'héritage ou des contraintes de cohérence, et de contrôler la validité des données par rapport à ces types ; JSON permet le typage des données mais peu complexe (voir JSON Schema).
- XML permet l'usage d'espaces de nom (namespaces) qui autorisent l'usage combiné de plusieurs langages XML différents utilisant un vocabulaire commun ; JSON ne le permet pas.

- Les documents XML sont auto documentés ; on sait à quel schéma ils doivent se conformer. En JSON, c'est le code appelant qui met en regard instance JSON et schema JSON pour s'assurer de la validité.

- La lecture directe, par le programmeur ou un utilisateur, d'un fichier XML est (en général) plus intelligible que celle d'un fichier JSON.

3 Pourquoi C# .Net ?

Le choix d'un langage particulier pour ce module a été compliqué. Il résulte de multiples contraintes. Il fallait que :

- le langage soit objet et assez proche de Java (pour lequel vous avez des cours)
- le langage soit suffisamment moderne pour être employé en entreprise
- le langage évite la lourdeur de l'apprentissage d'un langage très complexe comme C++ moderne
- le typage dans ce langage soit plus avancé que celui de Java
- le langage dispose d'une API performante permettant d'utiliser XML
- le langage puisse permettre une écriture dans un style fonctionnel, donc puisse correctement prendre en compte les critères nécessaires pour une écriture en langage fonctionnel : immuabilité, fonctions pures, fonctions d'ordre supérieur, notion de foncteur ... L'alternative étant l'usage de Java ou C++ combinée à un langage de programmation fonctionnelle comme Haskell, Purescript, Scheme ou Racket (du Lisp).
- le langage soit multi-plateforme
- le langage dispose d'une bibliothèque permettant le jeu vidéo (pour le projet)

Le langage C# (C Sharp) appartenant au framework DotNet développé par Microsoft, répond à toutes ces contraintes. C'est un langage moderne qui permet une écriture dans un style très élégant. Par dessus tout, dans ce même framework DotNet existe un langage nommé F# qui est un langage de programmation fonctionnelle pur. L'intérêt du framework DotNet est de permettra l'inter-opérabilité de ces langages : il est tout à fait possible de mélanger du code F# dans du C# !!!

4 Interface de Développement Intégrée

Parmi les IDE disponibles pour C#, celles qui ressortent sont :

- VS Code (sous Windows, Linux et Mac)
- Visual Studio (Windows seulement)
- Jetbrains Rider

C'est ce dernier que nous utiliserons. Je déconseille fermement l'usage de VS Code. VS Code n'est pas à proprement dit un IDE, mais seulement un éditeur de texte-code avancé.

Jetbrains Rider est une solution non publique mais qui autorise un usage gratuit pour les universitaires et étudiants. Vous trouverez la procédure d'obtention d'une licence et d'installation ici : Licence et installation Jetbrains Nous utiliserons Monogame pour écrire le jeu vidéo qui constituera votre projet à rendre. Monogame est un framework et non un moteur de jeu comme Unity ou Unreal. Il permet cependant de gérer l'affichage d'objets, de gérer une boucle de jeu et ses callbacks, de gerer les contrôles (clavier, souris ...), etc.

Nous verrons son usage en TP mais vous trouverez ici un bon tutoriel, suffisamment simple et court pour rapidement utiliser Monogame : Tutoriel Monogame [video] de la chaîne Coding With Sphere.

Le site de référence de Monogame vous permettra d'installer le framework. Vous y trouverez également des tutoriels et la doc officielle.

5 Bibliothèque ReactiveX

Vers la fin du projet, nous nous essayerons à la programmation réactive (utilisant la programmation fonctionnelle) en C#, via le framework ReactiveX. Il vous faudra l'installer (cloner le dépot github).

Vous trouverez un exemple visuel d'application de ReactiveX ici : Marbles in RX

Côté tutos, je vous renvoie vers 2 sites :

- Introduction to RX
- Introducing RXJava

Le deuxième tuto traite de RXJava, mais l'API est très proche de RXC#.

6 Contrat Pédagogique

Le cours de *Formalisation des Connaissances et Programmation Fonctionnelle* a une forme un peu particulière. Il est composé d'un **travail préparatoire** que vous devez faire **AVANT** de venir aux sessions de Cours-TD (CTD).

En effet, chaque semaine, il vous sera demandé d'étudier (pas seulement lire rapidement) le cours pour le prochain CTD. Vous devez, pour chaque CTD:

1. **Lire et Apprendre** les chapitres de cours fournis au format pdf.
2. **Écouter / Regarder / Comprendre / Connaître** les éventuels exemples / animations fournis avec le cours
3. **Vérifier vos connaissances** en faisant l'exercice corrigé du polycopié de TD (sans regarder la correction avant de l'avoir fait) et la vérification de vos réponses
4. **Préparer** d'éventuelles questions à poser en CTD

7 Horaires et Organisation

La promotion est divisée en plusieurs groupes équilibrés en effectif. Il est éventuellement possible pour un étudiant de changer de groupe, mais il doit trouver un camarade de l'autre groupe qui veut bien échanger avec lui.
Les dates et horaires sont données à la rentrée et sur le site du cours.

8 Évaluation

La note finale de l'UE *Formalisation des données* sera composée

- pour 60% de la note de l'examen final
- pour 40% de la note de contrôle continu, elle-même composée de
  - de la notes d'éventuels QCM faits en début de CTD
  - de la note du Projet Jeu (rendu code C# + XML/XSD/... + rapport + présentation)
  - de la note du Projet Cabinet Infirmier (rendu code C# + XML/XSD/... + rapport + présentation)