1 Seminarios de C# *(Primera Parte)*

Los requerimientos de cada ejercicio del seminario serán expuestos desde el punto de vista práctico y teórico; es decir, para su exposición, cada equipo se basará en el caso práctico en cuestión para introducir y explicar el elemento teórico requerido. La exposición no es una mera enunciación de código. Preguntas como: *¿Por qué?*, *¿Basándose en qué?*, *¿Cómo se logra esto en el lenguaje* `X`*?* entre otras, deben hacerse.

Todos los miembros del equipo deben participar en la solución del ejercicio y estar preparados para exponer todo el trabajo. **La persona a exponer** se decide el día de la exposición. Quién no esté presente en la exposición de su equipo tiene `0` en la evaluación. (Note que estas notas se promedian y hay distinción entre 0 y 2).

2 Seminario 4 - DSL en C# 4.0

1. Un `DSL` interno (según Martin Fowler) es un lenguaje de dominio específico que se define a partir de un lenguaje de programación base mediante la utilización de sus características. Hay lenguajes de programación que brindan características específicas que facilitan la creación de pequeños `DSL`. Cuando el lenguaje posee una sintaxis poco flexible como `C#` y `Java`, es común que se utilice un patrón de diseño conocido como "*fluent interfaces*". Diseñe un DSL interno en `C#` 3.5 que permita la definición de personas con atributos: `FirstName`, `LastName`. Estos objetos deberán poderse crear de las siguientes formas:

```
// Accediendo directamente a los atributos
var p1 = Factory.New.Person;
p1.FirstName = "Louis";
p1.LastName = "Dejardin";

// Accediendo a los atributos en forma de diccionario
var p2 = Factory.New.Person;
p2["FirstName"] = "Louis";
p2["LastName"] = "Dejardin";

// Inicializando mediante una "fluent interface"
var p3 = Factory.New.Person.FirstName("Louis").LastName("Dejardin");
// Con notación similar a JSON
var p4 = Factory.New.Person(FirstName: "Louis", LastName: "Dejardin");
```

2. ¿Cómo se pudiera lograr (en `C#` 4.0) que en nuestro `DSL` los atributos de la persona

puedan definirse dinámicamente, es decir, poder inicializar una persona con *manager*, aun cuando no fuera concebido así en un principio? Implemente esta versión del `DSL` dinámico.

```
var person = Factory.New.Person(
    FirstName: "Louis",
    LastName: "Dejardin",
    Manager: New.Person(
        FirstName: "Bertrand",
        LastName: "Le Roy",
    );
);
```

3. Implemente la clase `Factory` de manera que pueda inicializar a través de su método `New` cualquier tipo conocido en ejecución. Investigue qué características de un LP favorecen la concepción de `DSL` embebidos.

Objetivos: Uso de DynamicObject para la creación de `DSL`, Fluent Programming, Named-Parameters, Reflection.