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).

1.1 Seminario 1 - Clausura

1. Ejecute el siguiente código y reponda: ¿Por qué el resultado que sale en pantalla no es el esperado? Explicar el concepto de clausura (*closure*) y la forma (antinatural) en que C# captura las variables en la clausura. Apoyar la explicación con el código `IL` generado (use `Reflector`).

```
var actions = new Action[10];
for (int x = 0; x < actions.Length; x++)
{
    int y = x;
    actions[x] = () =>
    {
        int z = x;
        Console.WriteLine("{0}, {1}, {2}\n", x, y, z);
    };
}
```

2. Explique por qué el siguiente código lanza excepción:

```
var actions = new List<Action>();
string[] urls =
{
    "http://www.url.com",
    "http://www.someurl.com",
    "http://www.someotherurl.com",
    "http://www.yetanotherurl.com"
};
```

```
for (int i = 0; i < urls.Length; i++)
{
    actions.Add(() => Console.WriteLine(urls[i]));
}

foreach (var action in actions)
{
    action();
}
```

3. Explique y diga qué imprime el siguiente código en C++, el cual refleja algunas características de las expresiones *lambda*:

```
auto funcs = vector<function<int()>>();

int x = 1;
funcs.push_back([=] { return x; });

x = 2;
funcs.push_back([&] { return x; });

x++;
funcs.push_back([x = 4] { return x; });

for (auto f : funcs)
{
    int y = f();
    cout << y << endl;
}
```

a. ¿Cómo se pudiera acceder a los miembros de una clase desde el ámbito de una expresión *lambda*?

4. ¿Existen delegados en Java? ¿Existen las expresiones *lambda*? ¿Qué es el *Strategy Pattern*? ¿Cómo se pudiera lograr en caso de no existir?