Seminario de Lenguajes (.NET)

Práctica 5

1) Codificar la clase **`Cuenta`** de tal forma que el siguiente código produzca la salida por consola que se indica.

```
...
Cuenta c1 = new Cuenta();
c1.Depositar(100).Depositar(50).Extraer(120).Extraer(50);
Cuenta c2 = new Cuenta();
c2.Depositar(200).Depositar(800);
new Cuenta().Depositar(20).Extraer(20);
c2.Extraer(1000).Extraer(1);
Console.WriteLine("\nDETALLE");
Cuenta.ImprimirDetalle();
...
```

Salida por consola

```
Se creó la cuenta Id=1
Se depositó 100 en la cuenta 1 (Saldo=100)
Se depositó 50 en la cuenta 1 (Saldo=150)
Se extrajo 120 de la cuenta 1 (Saldo=30)
Operación denegada - Saldo insuficiente
Se creó la cuenta Id=2
Se depositó 200 en la cuenta 2 (Saldo=200)
Se depositó 800 en la cuenta 2 (Saldo=1000)
Se creó la cuenta Id=3
Se depositó 20 en la cuenta 3 (Saldo=20)
Se extrajo 20 de la cuenta 3 (Saldo=0)
Se extrajo 1000 de la cuenta 2 (Saldo=0)
Operación denegada - Saldo insuficiente

DETALLE
CUENTAS CREADAS: 3
DEPÓSITOS:       5    - Total depositado:  1170
EXTRACCIONES:    3    - Total extraído:    1140
                      - Saldo                30
 * Se denegaron 2 extracciones por falta de fondos
```

2) Agregar a la clase **`Cuenta`** del ejercicio anterior un método estático **`GetCuentas()`** que devuelva un **`List<Cuenta>`** con todas las cuentas creadas. Controlar que la modificación de la lista devuelta, por ejemplo borrando algún elemento, no afecte el listado que internamente mantiene la clase **`Cuenta`**. Sin embargo debe ser posible interactuar efectivamente con los objetos **`Cuenta`** de la lista devuelta. Verificar que el siguiente código produzca la salida por consola que se indica:

```
new Cuenta();
new Cuenta();
List<Cuenta> cuentas = Cuenta.GetCuentas();
// se recuperó la lista de cuentas creadas

cuentas[0].Depositar(50);
// se depositó 50 en la primera cuenta de la lista devuelta

cuentas.RemoveAt(0);
Console.WriteLine(cuentas.Count);
// se borró un elemento de la lista devuelta
// pero la clase Cuenta sigue manteniendo todos
// los datos "La cuenta id: 1 tiene 50 de saldo"

cuentas = Cuenta.GetCuentas();
Console.WriteLine(cuentas.Count);
// se recupera nuevamente la lista de cuentas

cuentas[0].Extraer(30);
//se extrajo 30 de la cuenta id: 1 que tenía 50 de saldo
```

Salida por consola

```
Se creó la cuenta Id=1
Se creó la cuenta Id=2
Se depositó 50 en la cuenta 1 (Saldo=50)
1
2
Se extrajo 30 de la cuenta 1 (Saldo=20)
```

3) Reemplazar el método estático **`GetCuentas()`** del ejercicio anterior por una propiedad estática de sólo lectura.

4) Indicar en cada caso si la definición de la clase **A** es correcta, en caso de no serlo detallar cuál es el error.

a)

```
class A {
    static int s_a=0;
    static A() {
        s_a++;
    }
    public A() {
         s_a++;
    }
}
```

b)

```
class A {
    static int s_a = 0;
    public static A() {
        s_a++;
    }
    A() {
        s_a++;
    }
 }
```

c)

```
class A {
   static int s_a = 0;
   static A(int a) {
       s_a=a;
   }
   A(int a) {
       s_a = a;
   }
}
```

d)

```
class A {
    static int s_a = 0;
    int a = 0;
    static A() {
        s_a = 30;
    }
    A(int a) {
        s_a = a;
        this.a = a;
    }
}
```

e)

```
class A {
    static int s_a = 0;
    int a = 0;
    static A() {
        a = 30;
    }
    A(int a) {
        a = s_a;
    }
}
```

f)

```
class A
{
    static int s_a = 1;
    int a = 0;
    static A() => s_a += s_a;
    public static A GetInstancia()
                    => new A(1);
    A(int a) => this.a = a + s_a;
}
```

g)

```
class A
{
   const double PI = 3.1416;
   static double DoblePI = 2 * PI;
}
```

h)

```
class A
{
  static double PI = 3.1416;
  const double DoblePI = 2*PI;
}
```

i)

```
class A {
    static readonly List<int> _lista;
    static A() {
        _lista = new List<int>();
    }
    public static void P(int i) {
        _lista.Add(i);
    }
}
```

j)

```
class A {
  static readonly List<int> _lista;
  static A() {
          _lista = new List<int>();
  }
  public static void P(List<int> li) {
      _lista = li;
  }
}
```

k)

```
class A
{
   static int[] vector = { 1, 2, 3 };
   public int this[int i]
   {
      get { return vector[i]; }
   }
}
```

l)

```
class A
{
   static int[] vector = { 1, 2, 3 };
   public static int this[int i]
   {
       get { return vector[i]; }
   }
}
```

5) Qué líneas del código siguiente provocan error de compilación? Analizar cuándo es posible acceder a miembros estáticos y de instancia.

```
class A
{
    char c;
    static string st;
    void metodo1()
    {
        st = "string";
        c = 'A';
    }
    static void metodo2()
    {
        new ClaseA().c = 'a';
        st = "st2";
        c = 'B';
        new A().st = "otro string";
    }
}
```

6) Modificar la definición de la clase **`Matriz`** realizada en la práctica 4. Eliminar los métodos **`SetElemento(...)`** y **`GetElemento(...)`**. Definir un indizador adecuado para leer y escribir elementos de la matriz. Eliminar los métodos **`GetDiagonalPrincipal()`** y **`GetDiagonalSecundaria()`** reemplazándolos por las propiedades de sólo lectura **`DiagonalPrincipal`** y **`DiagonalSecundaria`**.

7) Definir la clase **`Persona`** con las siguientes propiedades de lectura y escritura: **`Nombre`** de tipo **`string`**, **`Sexo`** de tipo **`char`**, **`DNI`** de tipo **`int`**, y **`FechaNacimiento`** de tipo **`DateTime`**. Además definir una propiedad de sólo lectura (calculada) **`Edad`** de tipo **`int`**. Definir un indizador de lectura/escritura que permita acceder a las propiedades a través de un índice entero. Así, si **`p`** es un objeto **`Persona`**, con **`p[0]`** se accede al nombre, **`p[1]`** al sexo **`p[2]`** al DNI, **`p[3]`** a la fecha de nacimiento y **`p[4]`** a la edad. En caso de asignar **`p[4]`** simplemente el valor es descartado. Observar que el tipo del indizador debe ser capaz almacenar valores de tipo **`int`**, **`char`**, **`DateTime`** y **`string`**.

8) Dada la siguiente definición incompleta de clase:

```
class ListaDePersonas
{
    public void Agregar(Persona p)
    {
        . . .
    }
    . . .
}
```

Completarla y agregar dos indizadores de sólo lectura
Un índice entero que permite acceder a las personas de la lista por número de documento. Por ejemplo `p=lista[30456345]` devuelve el objeto `Persona` que tiene DNI=30456345 o `null` en caso que no exista en la lista.
Un índice de tipo `char` que devuelve un `List<string>` con todos los nombres de las personas de la lista que comienzan con el carácter pasado como índice.

9) ¿Cuál es el error en el siguiente programa?

```
Auto a = new Auto();
a.Marca = "Ford";
Console.WriteLine(a.Marca);

class Auto
{
   private string marca;
   public string Marca
   {
       set
       {
           Marca = value;
       }
       get
       {
           return marca;
       }
   }
}
```

Nota: Observar que utilizar la convención de prefijar a los campos privados con guión bajo, hace más difícil cometer este tipo de errores

10) Identificar todos los miembros en la siguiente declaración de clase. Indicar si se trata de un constructor, método, campo, propiedad o indizador, si es estático o de instancia, y en caso que corresponda si es de sólo lectura, sólo escritura o lectura y escritura. En el caso de las propiedades indicar también si se trata de una propiedad auto-implementada.
Nota: La clase compila perfectamente. Sólo prestar atención a la sintaxis, la semántica es irrelevante.

```
class A
{
    private static int a;
    private static readonly int b;
    A() { }
    public A(int i) : this() { }
    static A() => b = 2;
    int c;
    public static void A1() => a = 1;
    public int A1(int a) => A.a + a;
    public static int A2;
    static int A3 => 3;
    private int A4() => 4;
    public int A5 { get => 5; }
    int A6 { set => c = value; }
    public int A7 { get; set; }
    public int A8 { get; } = 8;
    public int this[int i] => i;
}
```

11) ¿ Qué diferencia hay entre estas dos declaraciones?

```
a)      public int X = 3;       y    b)      public int X => 3;
```