Seminario de Lenguajes (.NET)

Práctica 6

1) Sin borrar ni modificar ninguna línea, completar la definición de las clases **B**, **C** y **D**

```
class A
{
    protected int _id;
    public A(int id) => _id = id;
    public virtual void Imprimir() => Console.WriteLine($"A_{_id}");
}
class B : A
{
    . . .
}
class C : B
{
    . . .
}
class D : C
{
    . . .
    public override void Imprimir()
    {
        . . .
        base.Imprimir();
    }
}
```

para que el siguiente código produzca la salida indicada:

```
A[] vector = new A[] { new A(3), new B(5), new C(15), new D(41) };
foreach (A a in vector)
{
    a.Imprimir();
}
```

Salida por consola

```
A_3
B_5 --> A_5
C_15 --> B_15 --> A_15
D_41 --> C_41 --> B_41 --> A_41
```

2) Aunque consultar en el código por el tipo de un objeto indica habitualmente un diseño ineficiente, por motivos didácticos vamos a utilizarlo. Completar el siguiente código, que utiliza las clases definidas en el ejercicio anterior, para que se produzca la salida indicada:

```
A[] vector = new A[] { new C(1), new D(2), new B(3), new D(4), new B(5) };
foreach (A a in vector)
{
   ...
}
```

Salida por consola

```
B_3 --> A_3
B_5 --> A_5
```

Es decir, se deben imprimir sólo los objetos cuyo tipo exacto sea **`B`**

a) Utilizando el operador **`is`**

b) Utilizando el método **`GetType()`** y el operador **`typeof()`** (investigar sobre éste último en la documentación en línea de .net)

3) ¿Por qué no funciona el siguiente código? ¿Cómo se puede solucionar fácilmente?

```
class Auto
{
    double velocidad;
    public virtual void Acelerar()
        => Console.WriteLine("Velocidad = {0}", velocidad += 10);
}

class Taxi : Auto
{
    public override void Acelerar()
        => Console.WriteLine("Velocidad = {0}", velocidad += 5);
}
```

4) Contestar sobre el siguiente programa:

```
Taxi t = new Taxi(3);
Console.WriteLine($"Un {t.Marca} con {t.Pasajeros} pasajeros");

class Auto
{
   public string Marca { get; private set; } = "Ford";
   public Auto(string marca) => this.Marca = marca;
   public Auto() { }
}

class Taxi : Auto
{
   public int Pasajeros { get; private set; }
   public Taxi(int pasajeros) => this.Pasajeros = pasajeros;
}
```

¿Por qué no es necesario agregar **`:base`** en el constructor de **`Taxi`**? Eliminar el segundo constructor de la clase **`Auto`** y modificar la clase **`Taxi`** para el programa siga funcionando

5) ¿Qué líneas del siguiente código provocan error de compilación y por qué?

```
class Persona
{
    public string Nombre { get; set; }
}

public class Auto
{
    private Persona _dueño1, _dueño2;
    public Persona GetPrimerDueño() => _dueño1;
    protected Persona SegundoDueño
    {
        set => _dueño2 = value;
    }
}
```

6) Señalar el error en cada uno de los siguientes casos:

(6.1)

```
class A
{
    public string M1()
       => "A.M1";
}
class B : A
{
    public override string M1()
       => "B.M1";
}
```

(6.2)

```
class A
{
    public abstract string M1();
}

class B : A
{
    public override string M1()
       => "B.M1";
}
```

(6.3)

```
abstract class A
{
    public abstract string M1()
       => "A.M1";
}
class B : A
{
    public override string M1()
       => "B.M1";
}
```

(6.4)

```
class A
{
    public override string M1()
       => "A.M1";
}
class B : A
{
    public override string M1()
       => "B.M1";
}
```

(6.5)

```
class A
{
    public virtual string M1()
       => "A.M1";
}
class B : A
{
    protected override string M1()
       => "B.M1";
}
```

(6.6)

```
class A
{
   public static virtual string M1()
       => "A.M1";
}
class B : A
{
   public static override string M1()
       => "B.M1";
}
```

(6.7)

```
class A
{
   virtual string M1() => "A.M1";
}
class B : A
{
   override string M1() => "B.M1";
}
```

(6.8)

```
class A
{
    protected A(int i) { }
}
class B : A
{
    B() { }
}
```

(6.9)

```
class A
{
    private int _id;
    protected A(int i) => _id = i;
}
class B : A
{
    B(int i):base(5) {
        _id=i;
     }
}
```

(6.10)

```
class A
{
    private int Prop
    {
        set; public get;
    }
}
class B : A
{
}
```

(6.11)

```
abstract class A
{
    public abstract int Prop
       {set; get;}
}
class B : A
{
    public override int Prop
    {
        get => 5;
    }
}
```

(6.12)

```
abstract class A
{
    public int Prop {set; get;}
}
class B : A
{
    public override int Prop {
        get => 5;
        set {}
    }
}
```

7) Ofrecer una implementación polimórfica para mejorar el siguiente programa:

```
Imprimidor.Imprimir(new A(), new B(), new C(), new D());

class A {
    public void ImprimirA() => Console.WriteLine("Soy una instancia A");
}

class B {
    public void ImprimirB() => Console.WriteLine("Soy una instancia B");
}

class C {
    public void ImprimirC() => Console.WriteLine("Soy una instancia C");
}

class D {
    public void ImprimirD() => Console.WriteLine("Soy una instancia D");
}

static class Imprimidor {
    public static void Imprimir(params object[] vector) {
        foreach (object o in vector) {
            if (o is A) { (o as A)?.ImprimirA(); }
            else if (o is B) { (o as B)?.ImprimirB(); }
            else if (o is C) { (o as C)?.ImprimirC(); }
            else if (o is D) { (o as D)?.ImprimirD(); }
        }
    }
}
```

8) Crear un programa para gestionar empleados en una empresa. Los empleados deben tener las propiedades públicas de sólo lectura **`Nombre`**, **`DNI`**, **`FechaDeIngreso`**, **`SalarioBase`** y **`Salario`**. Los valores de estas propiedades (a excepción de **`Salario`** que es una propiedad calculada) deben establecerse por medio de un constructor adecuado.
Existen dos tipos de empleados: **`Administrativo`** y **`Vendedor`**. No se podrán crear objetos de la clase padre **`Empleado`**, pero sí de sus clases hijas (**`Administrativo`** y **`Vendedor`**). Aparte de las propiedades de solo lectura mencionadas, el administrativo tiene otra propiedad pública de lectura/escritura llamada **`Premio`** y el vendedor tiene otra propiedad pública de lectura/escritura llamada **`Comision`**.
La propiedad de solo lectura **`Salario`**, se calcula como el salario base más la comisión o el premio según corresponda.
Las clases tendrán además un método público llamado **`AumentarSalario()`** que tendrá una implementación distinta en cada clase. En el caso del administrativo se incrementará el salario base en un 1% por cada año de antigüedad que posea en la empresa, en el caso del vendedor se incrementará el salario base en un 5% si su antigüedad es inferior a 10 años o en un 10% en caso contrario.
El siguiente código (ejecutado el día 9/4/2022) debería mostrar en la consola el resultado indicado:

```
Empleado[] empleados = new Empleado[] {
  new Administrativo("Ana", 20000000, DateTime.Parse("26/4/2018"), 10000) {Premio=1000},
  new Vendedor("Diego", 30000000, DateTime.Parse("2/4/2010"), 10000) {Comision=2000},
  new Vendedor("Luis", 33333333, DateTime.Parse("30/12/2011"), 10000) {Comision=2000}
};
foreach (Empleado e in empleados)
{
   Console.WriteLine(e);
   e.AumentarSalario();
   Console.WriteLine(e);
}
```

Salida por consola

```
Administrativo Nombre: Ana, DNI: 20000000 Antigüedad: 3
Salario base: 10000, Salario: 11000
------------
Administrativo Nombre: Ana, DNI: 20000000 Antigüedad: 3
Salario base: 10300, Salario: 11300
------------
Vendedor Nombre: Diego, DNI: 30000000 Antigüedad: 12
Salario base: 10000, Salario: 12000
------------
Vendedor Nombre: Diego, DNI: 30000000 Antigüedad: 12
Salario base: 11000, Salario: 13000
------------
Vendedor Nombre: Luis, DNI: 33333333 Antigüedad: 10
Salario base: 10000, Salario: 12000
------------
Vendedor Nombre: Luis, DNI: 33333333 Antigüedad: 10
Salario base: 11000, Salario: 13000
------------
```

Recomendaciones: Observar que el método `AumentarSalario()` y la propiedad de solo lectura `Salario` en la clase `Empleado` pueden declararse como abstractos. Intentar no utilizar campos sino propiedades auto-implementadas todas las veces que sea posible. Además sería deseable que la propiedad `SalarioBase` definida en `Empleado` sea pública para la lectura y protegida para la escritura, para que pueda establecerse desde las subclases `Administrativo` y `Vendedor`.