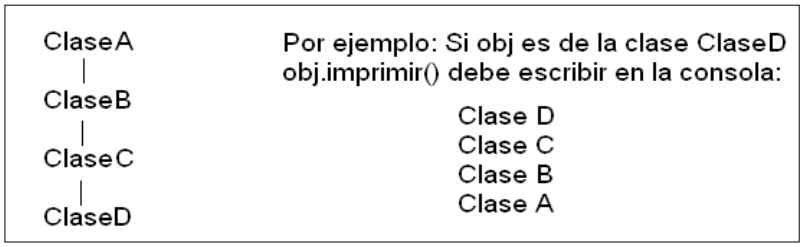
Seminario de Lenguajes (.NET)

Práctica 6

Ejercicio obligatorio para el coloquio: El **ejercicio 8** de esta práctica forma parte del conjunto de ejercicios de programación obligatorios que el alumno debe resolver y exponer de manera oral sobre máquina el día del coloquio hacia final de la cursada.

1) Cree una jerarquía de clases como la que se indica en el esquema y defina en todas ellas el método **`imprimir()`** que escribe en la consola la jerarquía desde la **`claseA`** hasta aquella a la que pertenece el objeto en forma invertida.



2) Codifique un programa que cree un arreglo de objetos **`ClaseA`** que contenga objetos **`ClaseC`**, y objetos **`ClaseD`**. Recorra el arreglo llamando al método **`imprimir()`** sólo para los objetos **`ClaseC`**.

3) Conteste las siguientes preguntas:

a) ¿Para qué sirve un constructor?

b) ¿Para qué sirve un destructor?

c) ¿Cuál es la diferencia entre **`Protected`** y **`Private`**?

d) ¿Para qué sirve invocar el método **`collect`** de la clase **`GC`**?

4) ¿Por qué no funciona el siguiente código? Cómo puede solucionarlo.

```
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);
   }
}
```

5) Utilizando las clases definidas en el ejercicio 4 codifique el siguiente programa:

```
class Ejercicio5{
   static void Main(){
      Auto a=new Auto();
      check(a);
      a=new Taxi();
      check(a);
      System.Console.ReadLine();
   }
   static void check(Auto a){
      if (a is Auto){
         Console.WriteLine("Es un auto no es un taxi");
      }else{
         if (a is Taxi) Console.WriteLine("Es un taxi");
      }
   }
}
```

¿Por qué en la segunda invocación del método **check** no se imprimió "Es un taxi"? ¿Cómo puede solucionarlo?

6) ¿Qué puede decir acerca de la definición de la clase **Taxi**? ¿Es necesario definirle un constructor?

```
class Auto{

  private string marca;

  public Auto(string marca){
    this.marca=marca;
  }
}

class Taxi:Auto{
}
```

7) Pruebe el siguiente programa. Preste atención a los constructores. ¿Porqué no es necesario agregar `:base` en el constructor de `Taxi`? Pruebe eliminar el segundo constructor de la clase `Auto` y observe lo que sucede.

```
using System;
    class Program{
      static void Main(){
        Taxi t=new Taxi(3);
        Console.WriteLine(t.marca+ t.pasajeros);
      }
    }
    class Auto{
      public string marca="Ford";
      public Auto(string marca){
        this.marca=marca;
      }
      public Auto(){  }
    }
    class Taxi:Auto{
      public int pasajeros;
      public Taxi(int pasajeros){
        this.pasajeros=pasajeros;
      }
    }
```

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.
En el método **Main()** de **Program** crear una cierta cantidad de empleados de distintos tipos, establecer las propiedades **Premio** o **Comisión** según corresponda, agregarlos a un vector de empleados y codifique lo siguiente:

```
foreach(Empleado e in empleados)
{
    Console.WriteLine(e.Salario);
    e.AumentarSalario();
    Console.WriteLine(e.Salario);
}
```

Recomendaciones: Observar que el método **AumentaSalario()** y la propiedad de solo lectura **Salario** en la clase **Empleado** pueden declararse como abstractos. Además sería deseable que todos los campos se declaren como privados y 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**.

9) Redefina el método **ToString()** de las clases **Vendedor** y **Administrativo** del ejercicio anterior devolviendo el contenido de todos sus campos privados. Verifique el resultado haciendo:

```
foreach(Empleado e in empleados) Console.WriteLine(e);
```