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
```

```
using System;
using System.Collections.Generic;

public class Cuenta
{
    private static int contadorCuentas = 0;
    private int id;
    private double saldo;
    private List<string> operaciones;

    public Cuenta()
    {
        this.id = ++contadorCuentas;
```

```
        this.saldo = 0;
        this.operaciones = new List<string>();
        Console.WriteLine("Se creó la cuenta Id={0}", this.id);
    }

    public Cuenta Depositar(double monto)
    {
        this.saldo += monto;
        this.operaciones.Add(string.Format("Se depositó {0} en la cuenta
{1} (Saldo={2})", monto, this.id, this.saldo));
        return this;
    }

    public Cuenta Extraer(double monto)
    {
        if (monto > this.saldo)
        {
            this.operaciones.Add("Operación denegada - Saldo
insuficiente");
            Console.WriteLine("Operación denegada - Saldo insuficiente");
        }
        else
        {
            this.saldo -= monto;
            this.operaciones.Add(string.Format("Se extrajo {0} de la cuenta
{1} (Saldo={2})", monto, this.id, this.saldo));
        }
        return this;
    }

    public static void ImprimirDetalle()
    {
        int cuentasCreadas = contadorCuentas;
        int totalDepositos = 0;
        int totalExtracciones = 0;
        int operacionesDenegadas = 0;
        foreach (Cuenta cuenta in cuentas)
        {
            foreach (string operacion in cuenta.operaciones)
            {
                if (operacion.StartsWith("Se depositó"))
```

```
                {
                    totalDepositos += int.Parse(operacion.Split(' ')[2]);
                }
                else if (operacion.StartsWith("Se extrajo"))
                {
                    totalExtracciones += int.Parse(operacion.Split('
')[2]);
                }
                else if (operacion.StartsWith("Operación denegada"))
                {
                    operacionesDenegadas++;
                }
                Console.WriteLine(operacion);
            }
        }
        Console.WriteLine("DETALLE");
        Console.WriteLine("CUENTAS CREADAS: {0}", cuentasCreadas);
        Console.WriteLine("DEPÓSITOS: {0} - Total depositado: {1}",
totalDepositos, totalDepositos);
        Console.WriteLine("EXTRACCIONES: {0} - Total extraído: {1}",
totalExtracciones, totalExtracciones);
        Console.WriteLine("- Saldo {0}", cuentas[cuentas.Count - 1].saldo);
        Console.WriteLine("* Se denegaron {0} extracciones por falta de
fondos", operacionesDenegadas);
    }
}
```

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
```

```
public static List<Cuenta> GetCuentas() {
          List<Cuenta> cuentas = new List<Cuenta>();
          foreach (Cuenta cuenta in Cuentas) {
              cuentas.Add(new Cuenta {
                  Id = cuenta.Id,
                  Saldo = cuenta.Saldo,
                  DetalleMovimientos = new
  List<string>(cuenta.DetalleMovimientos)
              });
          }
          return cuentas;
      }
```

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++;
    }
 }
```

a) La definición de la clase A es **correcta**. La clase A tiene un campo estático s_a y un constructor estático que se encarga de incrementar en 1 el valor de s_a cuando se carga la clase.

b) La definición de la clase A **no es correcta**. Hay un error en el constructor estático de la clase.

En la línea que define el constructor estático public static A(), se utiliza el nombre de la clase como si fuera el nombre del constructor. El constructor estático no tiene un nombre, solo la palabra clave static. Para corregir esto, la definición del constructor estático debería ser static A().

```
class A {
    static int s_a = 0;
    static A() {
        // Constructor estático para inicializar el campo estático s_a
        s_a = 0;
    }
    public A() {
        // Constructor de instancia para incrementar s_a en 1
        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;
    }
}
```

c) La definición de la clase A es **incorrecta.** Hay un error en la definición de los constructores.

En la definición de la clase, se definen dos constructores, uno estático y otro de instancia. Los constructores se llaman igual que la clase y, en consecuencia, son constructores. Sin embargo, ambos constructores tienen un parámetro int a, lo que significa que no están construyendo una instancia de la clase, sino que se espera que se llame al constructor con un argumento entero.

d) La definición de la clase A **no es correcta.** Hay dos problemas en el código:
El primer problema es que el constructor estático static A() no está definido correctamente. Un constructor estático no puede tener un modificador de acceso, ni siquiera public. Para corregir esto, se debe cambiar el nombre de los constructores, ya que los constructores deben tener el mismo nombre que la clase. Por ejemplo, se podría llamar al constructor estático static A.Initialize(int a) y al constructor de instancia public A(int a).
El segundo problema es que los dos constructores tienen el mismo nombre y el mismo parámetro, por lo que la sobrecarga no funciona. Debe haber una diferencia en el número o tipo de parámetros para que el compilador pueda distinguir entre ellos.

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;
}
```

e) La definición de la clase A **no es correcta**. Hay dos errores:

El método estático A() no tiene un tipo de retorno definido, lo cual es un error de sintaxis. Si se intenta definir un constructor estático, se debe usar el nombre de la clase en lugar de un tipo de retorno.

En el constructor de instancia A(int a), la asignación a = s_a se hace al revés. Debería ser s_a = a para asignar el valor del parámetro a la variable estática s_a.

f) La definición de la clase A **es correcta.**

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;
}
```

g) La definición de la clase A es **correcta**. La clase define una constante PI con valor 3.1416 y una variable estática DoblePI inicializada como el doble de la constante PI.

h) La definición de la clase A es **incorrecta.**
En la línea static double PI = 3.1416; se intenta inicializar una variable estática PI con el valor 3.1416. Sin embargo, no es posible inicializar una variable estática con una expresión que dependa del valor de otra variable, ya que las variables estáticas se inicializan antes de que se ejecute cualquier constructor o método de la clase.

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;
   }
}
```

i) La definición de la clase A es **correcta.** La clase A tiene un campo estático de solo lectura _lista, que se inicializa en el constructor estático de la clase. El método estático P agrega un entero a la lista _lista.

j) La definición de la clase A es **incorrecta**. La asignación directa a una variable readonly solo es posible en el constructor de la clase o en una inicialización estática en línea. En el caso de public static void P(List<int> li), se está intentando asignar un nuevo valor a _lista, lo que no es posible en una variable readonly.

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";
    }
}
```

Las lineas de codigo:
new ClaseA().c = 'a';
new A().st = "otro string";

En la línea **new ClaseA().c = 'a'**;, la clase ClaseA no está definida en ninguna parte del código proporcionado, por lo que el compilador no puede encontrarla y emite un error.

En la línea **new A().st = "otro string";**, se intenta acceder a un miembro estático st desde una instancia de la clase A. Es posible acceder a miembros estáticos a través de la clase misma, pero no a través de una instancia. Entonces, la línea debería ser A.st = "otro string";.

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**.

using System;

```
class Persona
{
    private string nombre;
    private char sexo;
    private int dni;
    private DateTime fechaNacimiento;

    public Persona(string nombre, char sexo, int dni, DateTime fechaNacimiento)
    {
        this.nombre = nombre;
        this.sexo = sexo;
        this.dni = dni;
        this.fechaNacimiento = fechaNacimiento;
    }

    public string Nombre
    {
        get { return nombre; }
        set { nombre = value; }
    }

    public char Sexo
    {
        get { return sexo; }
        set { sexo = value; }
    }

    public int DNI
    {
        get { return dni; }
        set { dni = value; }
    }

    public DateTime FechaNacimiento
    {
        get { return fechaNacimiento; }
        set { fechaNacimiento = value; }
    }

    public int Edad
    {
        get { return DateTime.Now.Year - fechaNacimiento.Year; }
    }

    public object this[int i]
    {
        get
        {
```

```
            switch (i)
            {
                case 0:
                    return nombre;
                case 1:
                    return sexo;
                case 2:
                    return dni;
                case 3:
                    return fechaNacimiento;
                case 4:
                    return Edad;
                default:
                    throw new IndexOutOfRangeException();
            }
        }
        set
        {
            switch (i)
            {
                case 0:
                    nombre = (string)value;
                    break;
                case 1:
                    sexo = (char)value;
                    break;
                case 2:
                    dni = (int)value;
                    break;
                case 3:
                    fechaNacimiento = (DateTime)value;
                    break;
                case 4:
                    // No se permite asignar la edad
                    break;
                default:
                    throw new IndexOutOfRangeException();
            }
        }
    }
}
```

La clase tiene las siguientes propiedades públicas de lectura y escritura: Nombre, Sexo, DNI y FechaNacimiento. La propiedad Edad es de solo lectura y se calcula automáticamente a partir de la fecha de nacimiento.

El indizador se implementa con la sintaxis this[int i], y permite acceder a las propiedades de la clase a través de un índice entero. El método get del indizador utiliza un switch para

seleccionar la propiedad correspondiente y devolver su valor, mientras que el método set actualiza la propiedad correspondiente en función del índice proporcionado.

El tipo de retorno del indizador es object para poder almacenar valores de diferentes tipos, como int, char, DateTime y string.