Descarter Redefinición de Operadores e Indizadores

Índice

Ejercicio 1

Diseñar la clase **Fraccion**, que representa el conjunto de los números racionales.
Un número racional se representa por un numerador, que es un número entero y un denominador, que es un número natural.
Esta clase debe ofrecer como mínimo los siguientes métodos públicos:

1. Constructor que recibe el numerado y el denominador y los simplifica.
2. Sobrescribe el método ToString(), para que devuelva una cadena con formato "num/den".
3. Redefinición del operador de cast implicito y explicito, para que devulevan el valor real de la fracción como double.
4. Propiedades para acceder y modificar el numerador y numerador simplificando la franción en caso que se modifique.
5. En todos los casos cuando el denominador al construir, al usar las propiedades, etc. sea cero. Generaremos una excepción DivideByZeroException con un mensaje indicándolo.
6. Redefiniremos las operaciones aritméticas simples, cuyo resultado será otra fracción en su forma simplificada:

$$Suma\ \ y\ \ resta => \frac{r_n}{r_d} = \frac{a_n * b_d + a_d * b_n}{a_d * b_d}$$

$$Multiplicación => \frac{r_n}{r_d} = \frac{a_n * b_n}{a_d * b_d}$$

$$División => \frac{r_n}{r_d} = \frac{a_n * b_d}{a_d * b_n}$$

7. Aplicación para probar el programa.

¿Cómo realizar la simplificación?

Para simplificar una fracción primero hay que hallar el máximo común divisor del numerador y del denominador.

Crearemos el método … private void simplifica()

Crearemos el método de clase privado private static int mcd(uint n, uint d) que se encargará de esta tarea y para ello, empleará el algoritmo de Euclides, cuyo funcionamiento se muestra en el siguiente ejemplo:

`Sea n = -1260 y d = 231`

1. Tomaremos el valor absoluto de **`n = Math.Abs(n)`**
2. En la primera iteración, se halla el resto **`r`** de dividir el primero **`n`** entre **`d`** . Se asigna a **`n`** el divisor **`d`** , y se asigna a **`d`** el resto **`r`** .
3. En la segunda iteración, se halla el resto **`r`** de dividir **`n`** entre **`d`** . Se asigna a **`n`** el divisor **`d`** , y se asigna a **`d`** el resto **`r`** .
4. Se repite el proceso hasta que el resto **`r`** sea cero.
5. El máximo común divisor será el último valor de **`d`** .

   `1260 = 231 * 5 + 105`

   `231 = 105 * 2 + 21`

   `105 = 21 * 5 + 0`
6. El máximo común divisor es 21.

$$\frac{-1260}{231} = \frac{\frac{1260}{21}}{\frac{231}{21}} = \frac{-60}{11}$$

Ejercicio2

Crea una clase **Hora** que a partir de una cadena del tipo "HH:MM" que llegará al constructor y usando expresiones regulares, descompondrá la información para almacenar los atributos hora y minutos.

Además, deberemos crear la clase **Horario** que implemente un indizador que reciba un objeto **Hora** que irá desde las 8:00 a las 13:00 horas y un valor enumerado con un día de la semana de Lunes a Viernes.
Ambos datos se solicitarán al usuario por teclado para mostrar una actividad usando el indizador. Ejemplo:

```
string actividad = horario[new Hora("9:45"), Dia.Lunes]
```

También redefiniremos el método ToString para mostrar todo un horario por pantalla. El horario se inicializará en un constructor por defecto con dicho indizador.

Ejercicio3

Vamos a practicar indizadores creando un tablero de ajedrez básico.
Para ello comenzaremos creando la clase **Pieza** con dos atributos enumerados, definidos en la clase: **Tipo** pieza con los valores {Peon, Caballo, Alfil, Torre, Dama, Rey}, **Color** con los valores {Bl, Nr}. Esta clase además tendrá el constructor para iniciar una pieza y el ToString que nos muestre la salida parecida a la siguiente:

```
Alfil-Bl
```

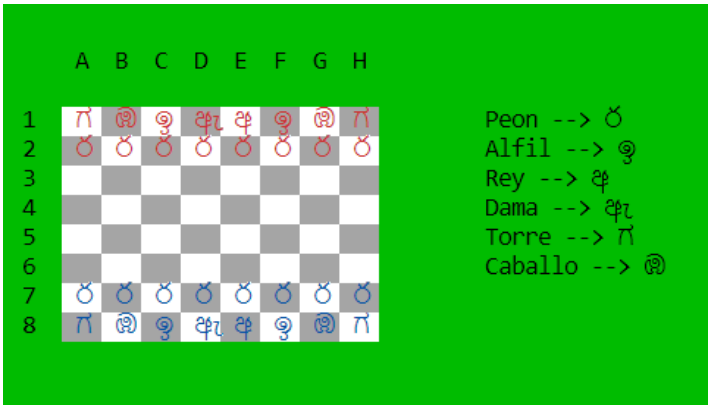
Cada pieza podrá estar posicionada en una casilla del tablero, por lo que tendremos que crear la clase **Casilla** formada por un **Color** y una Pieza. Crea el constructor necesario para asignar un color a la casilla, anula el ToString para mostrar la salida como la siguiente:

```
[  Alfil-Bl]
```

Crea las dos propiedades públicas para Pieza, de forma que el set controlará si la casilla está desocupada, en caso contrario lanzará excepción.
Por último crea la clase **Tablero** que estará formado por una matriz de Casillas, será inicializada en el constructor, aprovechando para asignar el color a las casillas.
Un indizador al que le llegue un string con la información de la casilla formato LetraNumero, ejemplo `A8` y se encargara de asignar la pieza o devolver la pieza en su caso (usarás expresiones regulares para decodificar la casilla a filas y columnas).
Otro indizado al que le llegará, directamente, la coordenada columna, fila para asignar o devolver la pieza.
Redefinición del ToString para que muestre el tablero.

Programa para inicializar un tablero, añadir algunas fichas y mostrar el tablero resultante.
Nota: si quieres obtener un resultado más vistoso, puedes usar los métodos que se pasan

más abajo y que permitirán mostrar un tablero semejante al siguiente. También se pasa el código de la main para inicializar el tablero y posicionar las fichas.



```
//En la clase Pieza
public void PintaPieza()
        {
            Encoding originalOutputEncoding = Console.OutputEncoding;
            Console.OutputEncoding = new UnicodeEncoding(!BitConverter.IsLittleEndian, false)
            string caracterPieza = tipo switch
            {
                Tipo.Peon => Char.ConvertFromUtf32(
                    (int)uint.Parse("c30", NumberStyles.HexNumber)),
                Tipo.Alfil => Char.ConvertFromUtf32(
                    (int)uint.Parse("d89", NumberStyles.HexNumber)),
                Tipo.Rey => Char.ConvertFromUtf32(
                    (int)uint.Parse("d85", NumberStyles.HexNumber)),
                Tipo.Dama => Char.ConvertFromUtf32((
                    int)uint.Parse("d87", NumberStyles.HexNumber)),
                Tipo.Torre => Char.ConvertFromUtf32(
                    (int)uint.Parse("c97", NumberStyles.HexNumber)),
                Tipo.Caballo => Char.ConvertFromUtf32(
                    (int)uint.Parse("db9", NumberStyles.HexNumber)),
            };
            Console.ForegroundColor = color == Color.Bl ? ConsoleColor.DarkBlue
                                                        : ConsoleColor.DarkRed;
            Console.Write(caracterPieza);
            Console.OutputEncoding = originalOutputEncoding;
        }
        public static void PintaPistaPiezas()
        {
            string[] codigoPiezas = new string[] { "c30", "d89", "d85", "d87",
                                                   "c97", "db9" };
            int fila = 4;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.BackgroundColor = ConsoleColor.DarkGreen;
            Encoding originalOutputEncoding = Console.OutputEncoding;
            Console.OutputEncoding = new UnicodeEncoding(!BitConverter.IsLittleEndian, false)
            for (int i = 0; i < codigoPiezas.Length; i++)
            {
                Console.SetCursorPosition(37, fila + i);
                Console.Write((Tipo)i + " --> " +
                 Char.ConvertFromUtf32((int)uint.Parse
                                       (codigoPiezas[i], NumberStyles.HexNumber)));
            }
            Console.SetCursorPosition(40, 20);
            Console.OutputEncoding = originalOutputEncoding;
        }

        //En la clase Tablero
         public void Muestra()
        {
            StringBuilder cadena = new StringBuilder("\n\n      ");
            Console.ForegroundColor = ConsoleColor.Black;
```

```
            for (char i = 'A'; i < 'A' + 8; i++) cadena.Append($"{i,-3}");
            cadena.Append("\n\n");
            Console.Write(cadena);
            for (int i = 0; i < Casillas.GetLength(0); i++)
            {
                Console.BackgroundColor = ConsoleColor.DarkGreen;
                Console.ForegroundColor = ConsoleColor.Black;
                Console.Write("  " + (i + 1) + "  ");
                for (int j = 0; j < Casillas.GetLength(1); j++)
                {
                    Console.BackgroundColor = Casillas[i, j].Color == Color.Bl ?
                                              ConsoleColor.White : ConsoleColor.Black;
                    Console.Write(" ");
                    if (Casillas[i, j].Pieza != null) Casillas[i, j].Pieza.PintaPieza();
                    else Console.Write(" ");
                    Console.Write(" ");
                }
                Console.Write("\n");
            }
            Pieza.PintaPistaPiezas();
        }
        //Main
        static void Main()
        {
            Console.BackgroundColor = ConsoleColor.DarkGreen;
            Console.Clear();
            Tablero tablero = new Tablero();
            tablero["A1"] = new Pieza(Pieza.Tipo.Torre, Color.Nr);
            tablero["B1"] = new Pieza(Pieza.Tipo.Caballo, Color.Nr);
            tablero["C1"] = new Pieza(Pieza.Tipo.Alfil, Color.Nr);
            tablero['d', 1] = new Pieza(Pieza.Tipo.Dama, Color.Nr);
            tablero["E1"] = new Pieza(Pieza.Tipo.Rey, Color.Nr);
            tablero["F1"] = new Pieza(Pieza.Tipo.Alfil, Color.Nr);
            tablero["G1"] = new Pieza(Pieza.Tipo.Caballo, Color.Nr);
            tablero['h', 1] = new Pieza(Pieza.Tipo.Torre, Color.Nr);
            tablero["A2"] = new Pieza(Pieza.Tipo.Peon, Color.Nr);
            tablero["b2"] = new Pieza(Pieza.Tipo.Peon, Color.Nr);
            tablero["C2"] = new Pieza(Pieza.Tipo.Peon, Color.Nr);
            tablero["D2"] = new Pieza(Pieza.Tipo.Peon, Color.Nr);
            tablero["e2"] = new Pieza(Pieza.Tipo.Peon, Color.Nr);
            tablero["f2"] = new Pieza(Pieza.Tipo.Peon, Color.Nr);
            tablero["g2"] = new Pieza(Pieza.Tipo.Peon, Color.Nr);
            tablero["h2"] = new Pieza(Pieza.Tipo.Peon, Color.Nr);

            tablero["A8"] = new Pieza(Pieza.Tipo.Torre, Color.Bl);
            tablero["B8"] = new Pieza(Pieza.Tipo.Caballo, Color.Bl);
            tablero["C8"] = new Pieza(Pieza.Tipo.Alfil, Color.Bl);
            tablero['d', 8] = new Pieza(Pieza.Tipo.Dama, Color.Bl);
            tablero["E8"] = new Pieza(Pieza.Tipo.Rey, Color.Bl);
            tablero["F8"] = new Pieza(Pieza.Tipo.Alfil, Color.Bl);
            tablero["G8"] = new Pieza(Pieza.Tipo.Caballo, Color.Bl);
```

```
            tablero['h', 8] = new Pieza(Pieza.Tipo.Torre, Color.Bl);
            tablero["A7"] = new Pieza(Pieza.Tipo.Peon, Color.Bl);
            tablero["b7"] = new Pieza(Pieza.Tipo.Peon, Color.Bl);
            tablero["C7"] = new Pieza(Pieza.Tipo.Peon, Color.Bl);
            tablero["D7"] = new Pieza(Pieza.Tipo.Peon, Color.Bl);
            tablero["e7"] = new Pieza(Pieza.Tipo.Peon, Color.Bl);
            tablero["f7"] = new Pieza(Pieza.Tipo.Peon, Color.Bl);
            tablero["g7"] = new Pieza(Pieza.Tipo.Peon, Color.Bl);
            tablero["h7"] = new Pieza(Pieza.Tipo.Peon, Color.Bl);
            tablero.Muestra();
            Console.ReadLine();
        }
```

Ejercicio 1

Para entender mejor los **Interfaces**, a continuación se muestra un ejemplo sencillo. Fíjate, sobre todo, en el programa principal para comprender mejor su utilidad.

```
interface IVisualiza
{
   void Visualiza();
}
class Triangulo : IVisualiza
{
   private double @base;
   private double altura;
   public Triangulo(double base_, double altura)
   {
       @base = base_;
       this.altura = altura;
   }
```

```
    private double area
    {
        get { return @base * altura / 2; }
    }
    public void Visualiza()
    {
        Console.WriteLine($"Base del triángulo: {@base}");
        Console.WriteLine($"Altura del triángulo: {altura}");
        Console.WriteLine($"Área del triángulo: {area}");
    }
}

class Proveedor : IVisualiza
{
    private string nombre;
    private string apellidos;
   public Proveedor(string nombre, string apellidos)
    {
        this.nombre = nombre;
        this.apellidos = apellidos;
    }
    public void Visualiza()
    {
        Console.WriteLine($"Nombre: {nombre}");
        Console.WriteLine($"Apellidos: {apellidos}");
    }
}
class EjemploInterfacesApp
{
    static void VerDatos(IVisualiza oVisualizable)
    {
        oVisualizable.Visualiza();
    }
    static void Main()
    {
        Triangulo t = new Triangulo(10, 5);
        VerDatos(t);
        Proveedor p = new Proveedor("Erik", "Erik otra vez");
        VerDatos(p);
    }
}
```

Nota: Crea una librería de interfaces llamada **MisInterfaces** a la que le irás añadiendo todas las interfaces que implementes en los ejercicios. Para poder usarla deberás incluirla en tus proyectos.

Ejercicio 6

Partiendo de la siguiente definición de clase parametrizada…

```
class A<T, U>
{
  private T clave;
  private U valor;
  …
}
```

- Define un constructor que reciba los dos atributos como parámetro.
- Crea 2 propiedades que te permitirán devolver los dos atributos.
- Prueba la clase en la Main con una clave de tipo entero y un valor de tipo cadena.

Nota: Sin usar el código *'autogenerado'* por el IDE.

Ejercicio 10

Crea una clase con un parámetro genérico **ParOrdenado** con dos atributos denominados primero y segundo del mismo tipo **T**. Crea el correspondiente constructor, propiedades y el método ToString.
Además, añade un **indizador** de solo lectura que dependiendo de si el índice es 0 te devolverá el valor de atributo primero y si es 1 el valor de atributo segundo (o una excepción en cualquier otro caso). Crea un programa que te permita probar esta clase usando enteros y cadenas.

✅ Ejercicio 11

A partir de la clase del ejercicio anterior, vamos a crear una clase **NumeroComplejo** que derivare de **ParOrdenado < double >**.
Como ya sabemos, los números complejos constan de dos partes una real y una imaginaria (compuesta por el número y el sufijo i), por eso vamos a utilizar la superclase ParOrdenado que ya posee los dos elementos que necesitamos.

- Tendremos un constructor al que le pasaremos un string con la forma binomial de un número complejo y se encargará de comprobarlo con una expresión regular.
- Además, también tendrás que redefinir los operadores +, - , *, ==, != de forma correcta y el cast explícito.
- Anula el método ToString que te devolverá el número con el sufijo y el signo + añadido.

Crea los elementos necesarios para que la clase quede completa y el programa para probar su funcionamiento.

Pista: Un número complejo se representa en forma binomial como:

$$z = a + bi$$

Las operaciones que se piden en el programa son…

$$Suma => (a, b) + (c, d) = (a + c, b + d)$$

$$Multiplicación => (a, b) \cdot (c, d) = (a \cdot c - b \cdot d, a \cdot d - c \cdot b)$$

$$Resta => (a, b) - (c, d) = (a - c, b - d)$$

$$Igualdad => (a, b) = (c, d) \Leftrightarrow a = c \wedge b = d$$