Enunciado:

Tema: **Miembros estáticos y de instancia, patrón Singleton**.

Suponga que existe una sola impresora para nuestra aplicación, sería un error poder crear más de un objeto del tipo impresora. Entonces *¿Cómo impedir que el equipo de desarrollo pueda crear varios objetos impresora cuando solo existe físicamente una*?, la respuesta a esta pregunta es el patrón de diseño conocido como **Singleton**. Este patrón prohíbe el uso de new ya que posee un constructor privado (25), y acumula la única instancia de la clase en un campo del tipo private static (13) tal como se ve en el código de la **Figura 1**.

Figura 1

```
11  public sealed class Impresora
12  {
13      private static Impresora _Instance;
14
15      public static Impresora Instance
16      {
17          get
18          {
19              if (_Instance == null)
20                  _Instance = new Impresora();
21              return _Instance;
22          }
23      }
24
25      private Impresora() { } //Constructor privado
26
27      public string Documento { get; set; }   //Documento a imprimir
28      public void Imprimir() { Console.WriteLine(this.Documento); }   //Imprime un documento
29  }
```

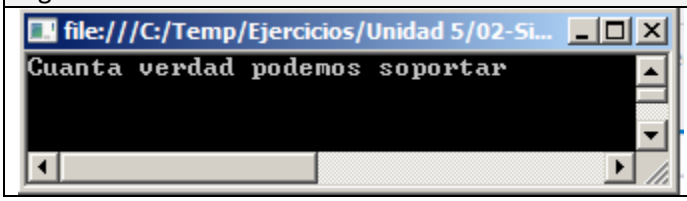
La línea de código (20) **Figura 1**, que crea la única instancia de la impresora se ejecuta a lo sumo una sola vez, ya que luego de ejecutarse, el campo "**_Instance**" se establece a new Impresora() y dejara de valer **null** y ya el flujo del programa no pasará más por la línea 20.

La **Figura 2**, muestra cómo podemos utilizar la única instancia de la impresora (12), luego se le establece el texto a imprimir (13) y se llama al método Imprimir() (14) para que imprima el texto, la salida se muestra en la **Figura 3**. Notemos que usar **new** para crear una nueva instancia de la impresora daría un error en tiempo de ejecución, la única manera de obtener una instancia es con la propiedad estática "Instance".

Figura 2

```
10  static void Main(string[] args)
11  {
12      Impresora impresora = Impresora.Instance;   //Se crea la única instancia de la impresora.
13      impresora.Documento = "Cuanta verdad podemos soportar";
14      impresora.Imprimir();
15
16      Console.ReadKey();
17  }
```

Figura 3

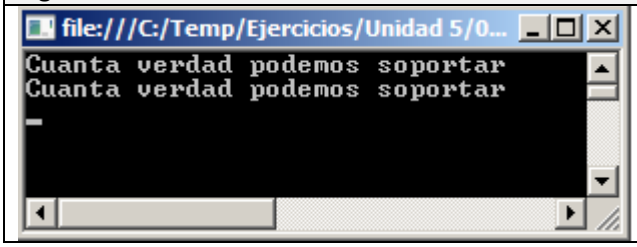


La **Figura 4** muestra otro posible Main(), en las líneas 14 y 17 se llama al método Imprimir() de dos objetos que imprimen exactamente el mismo mensaje, como podemos ver en la **Figura 5**, aunque nunca hemos establecido la propiedad **Documento** de la **impresora_A**, cosa que demuestra que son en realidad un mismo y único objeto, la única instancia de la impresora.

Figura 4

```
10  static void Main(string[] args)
11  {
12      Impresora impresora = Impresora.Instance;
13      impresora.Documento = "Cuanta verdad podemos soportar";
14      impresora.Imprimir();
15
16      Impresora impresora_A = Impresora.Instance;
17      impresora_A.Imprimir();
18
19      Console.ReadKey();
20  }
```

Figura 5



ReferenceEquals

La **Figura 6** muestra otro posible Main(), en el que se utiliza el método “ReferenceEquals” para demostrar que solo existe una instancia del tipo “Impresora”, tal como se ve en la línea 18, esta instancia esta referenciada por dos variable “impresora” e “impresora_A” que apuntan a la única instancia de la Impresora.

Figura 6

```
10 static void Main(string[] args)
11 {
12     Impresora impresora = Impresora.Instance;   //Se crea la única instancia de la impresora.
13     impresora.Documento = "Cuanta verdad podemos soportar";
14     impresora.Imprimir();
15
16     Impresora impresora_A = Impresora.Instance;
17
18     bool mismaImpresora = object.ReferenceEquals(impresora, impresora_A);
19     if (mismaImpresora)
20         Console.WriteLine("impresora e impresora_A son la misma y única impresora");
21
22     Console.ReadKey();
23 }
```

La **Figura 7**, muestra la salida que arroja la ejecución del código de la **Figura 6**.

Figura 7