ITR UTEC
Centro Sur
Universidad Tecnológica

TECNÓLOGO
EN INFORMÁTICA

Taller de sistemas de información .NET Tarea – 02_B

Autor: Lucas Techera

Docente: Gabriel Aramburu

CI: 5.295.981-3

Contacto: fidel.techera@estudiantes.utec.edu.uy

Consigna.

Refactoring del ejemplo planteado en clases.

Objetivos:

- Interiorizar el uso del estilo funcional en C#.
- Familiarizarse con librerías de C# para el manejo de estructuras de memoria.
- Familiarizarse con la estructura Map.

Se pide: El código de ejemplo planteado en clases implementa un pipeline que simula el procesamiento de requests: dado una request entrante se decide que endpoint la va a procesar.

Para lograr este comportamiento, cada endpoint se implementa como un paso concreto que se agrega al pipeline de ejecución.

Este diseño tiene una desventaja: hay que recorrer el pipeline cada vez que llega una request, orden de ejecución: O(n). Utilizando un Map, donde la clave sea la url y el valor sea una función, se puede optimizar el diseño realizado, ya que se accede a la función a ejecutar con un orden de ejecución de O(1)

Solución:

Para comenzar hay que aclarar que se comparará ambos casos de uso lista y Map, para luego eliminar el primero y refactorizar el código utilizando solo la estructura Map.

Vamos a definir una estructura de tipo Map para nuestros endpoints, en este caso como C# a diferencia de Java no cuenta con una implementación de Map propiamente dicha, se utilizan Diccionarios, como se muestra a continuación:

```
private Dictionary<string, Endpoint> _endpointsMap;
```

Inicializamos dicha estructura en el constructor de nuestra clase **PipeLineEstiloFuncional.cs** de la siguiente manera:

```
public PipeLineEstiloFuncional()
{
    _endpoints = new List<Endpoint>();
    _endpointsMap = new Dictionary<string , Endpoint>();
}
```

Como se encuentra implementado un código utilizando listas, se agregarán métodos nuevos, en lugar de refactorizarlos, esto con el fin de realizar una comparación, al final de esta guía, se refactorizará el código eliminando la implementación de las listas.

```
3 referencias
public void AgregarEndPoint(Endpoint nuevoEndpoint)
{
    _endpoints.Add(nuevoEndpoint);
}

3 referencias
public void AgregarEndPointMap(Endpoint nuevoEndpoint)
{

    string url = nuevoEndpoint.GetUrl(); //será la clave de mi objeto endpoint

    if (!_endpointsMap.ContainsKey(url))
    {
        _endpointsMap.Add(url, nuevoEndpoint);
    }
    else
    {
        Console.WriteLine("El endpoint ya se encuentra registrado");
    }
}
```

Se manejarán dos formas de agregar un endpoint, en el primero el mismo se agrega a una lista, y en el segundo a un Diccionario. Este último guarda los datos en formato clave-valor, y en este caso una url, corresponderá a un objeto endpoint que comparta dicha url.

```
public void ProcesarRequestEntrante(string urlRequestEntrate)
{
    Stopwatch stopwatch = new Stopwatch();

    stopwatch.Start();

    foreach (Endpoint nuevoEndpoint in _endpoints)
    {
        nuevoEndpoint.Evaluar(urlRequestEntrate);
    }

    stopwatch.Stop();

    Console.WriteLine("Tiempo medido en milisegundos: " + stopwatch.ElapsedMilliseconds.ToString());
    Console.WriteLine("Tiempo medido en ticks: " + stopwatch.ElapsedTicks.ToString());
}

1 referencia
public void ProcesarRequestEntranteMap(string urlRequestEntrate)
{
    Stopwatch stopwatch = new Stopwatch();
    stopwatch.Start();

    if (_endpointsMap.TryGetValue(urlRequestEntrate, out Endpoint endPoint))
    {
        endPoint.Evaluar(urlRequestEntrate);

        stopwatch.Stop();

        Console.WriteLine("Tiempo medido en milisegundos: " + stopwatch.ElapsedMilliseconds.ToString()); //imprimo tiempo de demora de la busqueda.
        Console.WriteLine("Tiempo medido en ticks: " + stopwatch.ElapsedTicks.ToString());
    }
    else
    {
        Console.WriteLine("No existe endpoint.");
    }

}
```

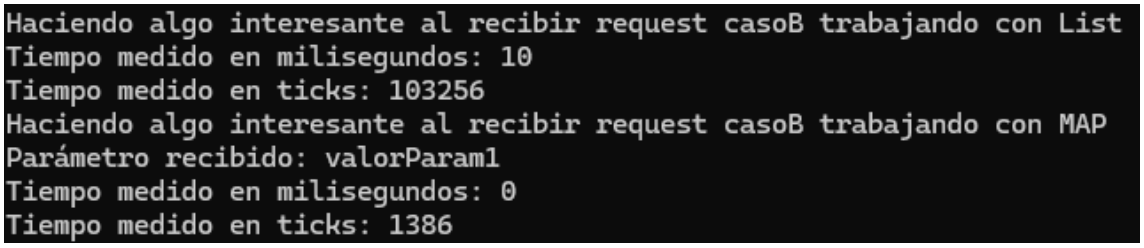
Al igual que en el caso anterior, para procesar peticiones también se tendrán ambas implementaciones, con el agregado respecto al código original de que ambas implementan la clase Stopwatch, con el fin de medir el tiempo de búsqueda entre ambas soluciones.

En el archivo **program.cs** se agregan nuevos datos para probar el pipeline modificado utilizando Map.

```
PipeLineEstiloFuncional pipeLine= new PipeLineEstiloFuncional();

pipeLine.AgregarEndPoint(new Endpoint("/casoA", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoA");
    Console.WriteLine($"Parámetro recibido: {s}");
}));


pipeLine.AgregarEndPoint(new Endpoint("/casoB", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoB");
}));

pipeLine.AgregarEndPoint(new Endpoint("/casoC", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoC");
}));

// cada vez que recibo un request ejecuto el pipeLine pasando como parámetro
// la urlDestino
pipeLine.ProcesarRequestEntrante("/casoB");

// Diseño utilizando map.

pipeLine.AgregarEndPointMap(new Endpoint("/casoA", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoA trabajando con MAP");
    Console.WriteLine($"Parámetro recibido: {s}");
}));

pipeLine.AgregarEndPointMap(new Endpoint("/casoB", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoB trabajando con MAP");
    Console.WriteLine($"Parámetro recibido: {s}");
}));

pipeLine.AgregarEndPointMap(new Endpoint("/casoC", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoC trabajando con MAP");
    Console.WriteLine($"Parámetro recibido: {s}");
}));

pipeLine.ProcesarRequestEntranteMap("/casoB");
```

A continuación, se procede a ejecutar el código para comprobar la diferencia entre ambos métodos.

Resultado:



Como se puede ver en la imagen el método que trabaja con List tomo 10 milisegundos, mientras que el método que trabaja con Map (Dictionary), tomó 0 milisegundos, es decir, fue demasiado rápido por lo que no llego a registrar tiempo, por lo que se tuvo que cambiar la medida para tener mayor precisión, y dado que la librería Stopwatch también nos proporciona una medición en ticks se utilizará la misma en adición a la anterior, basado en la misma se puede ver la abismal diferencia entre uno y otro. Hay que tener en cuenta que en este caso solo se prueba entre 3 endpoints ya que son ejemplos de prueba, pero en el mundo empresarial donde se manejan millones de peticiones, la diferencia es notoriamente grande.

A continuación, se muestra una ejecución, cuando el proyecto recién se abre ya que esto es mas pesado y se puede mostrar una variación respecto a las medidas anteriores.

```
Haciendo algo interesante al recibir request casoB trabajando con List
Tiempo medido en milisegundos: 28
Tiempo medido en ticks: 286192
Haciendo algo interesante al recibir request casoB trabajando con MAP
Parámetro recibido: valorParam1
Tiempo medido en milisegundos: 0
Tiempo medido en ticks: 1575
```

Se puede observar como un método toma casi el triple de tiempo registrado anteriormente, mientras que el otro apenas varia.

Ahora para refactorizar el código se procede a eliminar (comentar en este caso) la implementación de listas. Se adjuntan imágenes del código refactorizado.

PipeLineFuncional.cs:

```
namespace _01_02c_pipelineEstiloFuncional.pipeline
{

    3 referencias
    internal class PipeLineEstiloFuncional
    {
        //notese que por simplicidad aqui no estoy utilizando una interface
        //private IList<Endpoint> _endpoints; //lista de endpoints orden O(n)

        private Dictionary<string, Endpoint> _endpointsMap; //map (dictionary) de endpoints con orden de ejecución de O(1), en teoria mas rapido a la hora de buscar.

        1 referencia
        public PipeLineEstiloFuncional()
        {
            //_endpoints = new List<Endpoint>();
            _endpointsMap = new Dictionary<string , Endpoint>();
        }
        /*
        public void AgregarEndPoint(Endpoint nuevoEndpoint)
        {
            _endpoints.Add(nuevoEndpoint);
        }
        */
        3 referencias
        public void AgregarEndPointMap(Endpoint nuevoEndpoint)
        {

            string url = nuevoEndpoint.GetUrl(); //será la clave de mi objeto endpoint

            if (!_endpointsMap.ContainsKey(url))
            {
                _endpointsMap.Add(url, nuevoEndpoint);
            }
            else
            {
                Console.WriteLine("El endpoint ya se encuentra registrado");
            }
        }
```

```
/*
public void ProcesarRequestEntrante(string urlRequestEntrate)
{
    Stopwatch stopwatch = new Stopwatch();

    stopwatch.Start();

    foreach (Endpoint nuevoEndpoint in _endpoints)
    {
        nuevoEndpoint.Evaluar(urlRequestEntrate);
    }

    stopwatch.Stop();

    Console.WriteLine("Tiempo medido en milisegundos: " + stopwatch.ElapsedMilliseconds.ToString());
    Console.WriteLine("Tiempo medido en ticks: " + stopwatch.ElapsedTicks.ToString());
}
*/
1 referencia
public void ProcesarRequestEntranteMap(string urlRequestEntrate)
{
    Stopwatch stopwatch = new Stopwatch();
    stopwatch.Start();

    if (_endpointsMap.TryGetValue(urlRequestEntrate, out Endpoint endPoint))
    {
        endPoint.Evaluar(urlRequestEntrate);

        stopwatch.Stop();

        Console.WriteLine("Tiempo medido en milisegundos: " + stopwatch.ElapsedMilliseconds.ToString()); //imprimo tiempo de demora de la busqueda.
        Console.WriteLine("Tiempo medido en ticks: " + stopwatch.ElapsedTicks.ToString());
    }
    else
    {
        Console.WriteLine("No existe endpoint.");
    }

}
```

Program.cs:

```
PipeLineEstiloFuncional pipeLine= new PipeLineEstiloFuncional();

/*
pipeLine.AgregarEndPoint(new Endpoint("/casoA", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoA trabajando con List");
    Console.WriteLine($"Parámetro recibido: {s}");
}));


pipeLine.AgregarEndPoint(new Endpoint("/casoB", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoB trabajando con List");
}));

pipeLine.AgregarEndPoint(new Endpoint("/casoC", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoC trabajando con List");
}));

// cada vez que recibo un request ejecuto el pipeLine pasando como parámetro
// la urlDestino
pipeLine.ProcesarRequestEntrante("/casoB");
*/

// Diseño utilizando map.

pipeLine.AgregarEndPointMap(new Endpoint("/casoA", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoA trabajando con MAP");
    Console.WriteLine($"Parámetro recibido: {s}");
}));

pipeLine.AgregarEndPointMap(new Endpoint("/casoB", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoB trabajando con MAP");
    Console.WriteLine($"Parámetro recibido: {s}");
}));

pipeLine.AgregarEndPointMap(new Endpoint("/casoC", (string s) => {
    Console.WriteLine("Haciendo algo interesante al recibir request casoC trabajando con MAP");
    Console.WriteLine($"Parámetro recibido: {s}");
}));

pipeLine.ProcesarRequestEntranteMap("/casoB");
```

Y por último se realiza una ejecución con la implementación eliminada:

```
Haciendo algo interesante al recibir request casoB trabajando con MAP
Parámetro recibido: valorParam1
Tiempo medido en milisegundos: 10
Tiempo medido en ticks: 102660
```

Ahora se observa un incremento en el tiempo de búsqueda del método Map.

¿A qué se puede deber esto?

Hay diversos factores, una de ellas puede ser el compilador Just-In-Time, que al tener menos contexto para trabajar optimiza menos el código y puede demorar un poco mas o menos dependiendo del caso o la memoria caché que puede comportarse de manera diferente luego de la eliminación del primer método, si este ejemplo tuviera un tiempo de vida que dure hasta que se apague la aplicación podríamos notar como después de la primera ejecución los tiempos de carga regresarían a la normalidad.

A continuación, se deja un ejemplo, de un menor tiempo registrado.

```
Haciendo algo interesante al recibir request casoB trabajando con MAP
Parámetro recibido: valorParam1
Tiempo medido en milisegundos: 6
Tiempo medido en ticks: 62566
```

Repositorio con el código:
https://github.com/ElPiche/.NET_Excercices.git

Anexo:

https://www.c-sharpcorner.com/blogs/dictionary-and-maps-in-c-sharp

https://www.c-sharpcorner.com/UploadFile/9f4ff8/use-of-stopwatch-class-in-C-Sharp/