Ejercicio I01 - El relojero

Consigna

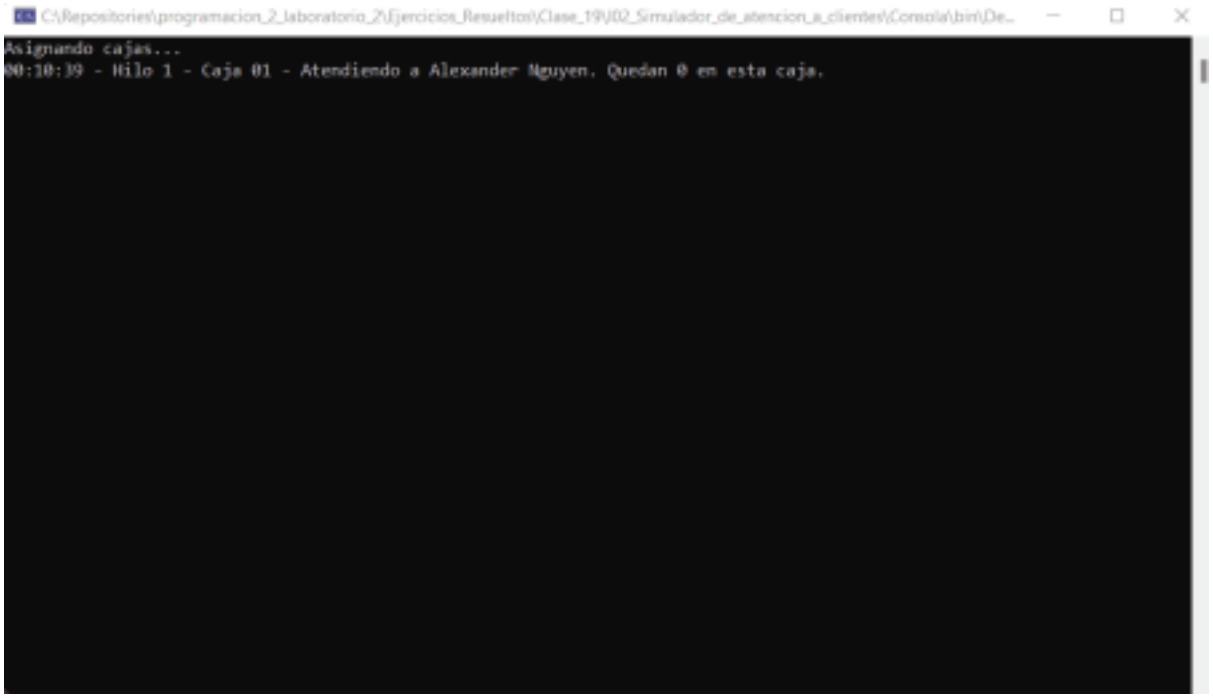
Crear un proyecto de Windows Forms con un `RichTextBox` y un `Label` dentro como se muestra en la siguiente imagen:

1. Crear el método `AsignarHora` que se encargará de imprimir la hora en el `Label lblHora`.
2. En el `Label` se deberá mostrar la fecha y hora actual, con segundos incluidos, y refrescándose una vez por segundo.
3. Generar tres prácticas, independientes, en el orden planteado: i. Realizar la actualización de la hora 1 vez por segundo utilizando alguna estructura iterativa dada en clase.
   ii. Agregar un control de Windows Forms de tipo `Timer`, configurarlo para refrescar la hora actual cada 1 segundo usando su evento `Tick`. iii. Resolver el mismo ejercicio utilizando programación multihilo.

Ejercicio I02 - Simulador de atención a clientes

Consigna

Se creará una solución para simular la atención paralela de clientes en dos cajas de un negocio.



Empezar creando un proyecto de biblioteca de clases y declarar las siguientes clases:

`Caja`

- Tendrá un atributo estático de tipo `Random` que será instanciado en un constructor estático.
- Tendrá un atributo de tipo `Queue<string>` llamado `clientesALaEspera`.
- Tendrá un atributo de tipo `string` llamado `nombreCaja` que será expuesto por una propiedad de sólo lectura.
- Tendrá una propiedad `CantidadDeClientesALaEspera` que retornará la cantidad actual de elementos en la cola `clientesALaEspera`. ● El único constructor de instancia deberá:
    - ○ Instanciar la cola `clientesALaEspera`.
    - ○ Recibir el nombre de la caja y asignarlo al atributo `nombreCaja`. ○ Recibir la referencia a un método que no retorne nada y reciba un objeto de tipo `Caja` y otro de tipo `string`. Dicha referencia deberá asignarse a un campo privado llamado `delegadoClienteAtendido`. Declarar un nuevo tipo delegado que permita cumplir con este punto. ● Tendrá un método `AgregarCliente` con visibilidad `internal` que recibirá un cliente y lo agregará a `clientesALaEspera`.
- Tendrá un método `IniciarAtencion` con visibilidad `internal` que deberá iniciar la atención de clientes en un sub-proceso paralelo. Este

método no recibirá nada y retornará la instancia de `Task` que se haya utilizado. ○ La tarea de atención de clientes:

- ■ Se seguirá ejecutando de manera iterativa hasta que se cierre la aplicación.
- ■ Si hay clientes a la espera (se puede verificar con el método Any de la biblioteca *LINQ* (`System.Linq`) que retorna `true` si una colección contiene algún elemento.):
  - ■ Se retirará al siguiente cliente de la cola `clientesALaEspera`.
  - ■ Se invocará al método referenciado por `delegadoClienteAtendido`, pasándole la instancia de esa misma caja y el cliente que se recuperó de la cola.
  - ■ Se suspenderá el hilo por un periodo de 1 a 5 segundos de manera aleatoria (usar atributo `Random`).

Negocio

- Requerirá la instalación del paquete NuGet `NameGenerator` en el proyecto de biblioteca de clases.
- Crear un atributo estático de tipo `RealNameGenerator` (namespace `NameGenerator.Generators`) que será instanciado en un constructor estático.
- Tendrá un atributo de tipo `ConcurrentQueue<string>` llamado `clientes`. Este tipo de dato es la versión del tipo `Queue` que cuenta con seguridad para trabajar con hilos (es *thread-safe*).
- Tendrá un atributo de tipo `List<Caja>` llamado `cajas`.
- El único constructor de instancia deberá:
  - ○ Recibirá una lista de cajas y la asignará al atributo `cajas`.
  - ○ Instanciará el atributo `clientes`.
- Tendrá un método público `ComenzarAtencion` que:
  - ○ Lo primero que hará será abrir todas las cajas del negocio, llamando al método `IniciarAtencion` de cada caja.
  - ○ En segundo lugar iniciará una tarea concurrente (en otro hilo) de simulación de clientes.
    - ■ La tarea se seguirá ejecutando de manera iterativa hasta que se cierre la aplicación.
    - ■ Esta tarea deberá agregar un nuevo cliente a la cola `clientes` cada un segundo (suspender 1 segundo el hilo).
    - ■ Para generar el nuevo cliente utilizar el método `Generate` del

atributo estático de tipo `RealNameGenerator`.

- En tercer lugar iniciará una tarea concurrente (en otro hilo) de asignación de cajas. Esta tarea asignará al siguiente cliente en la cola `clientes` a la caja con MENOS clientes .
  - La tarea se seguirá ejecutando de manera iterativa hasta que se cierre la aplicación.
  - Para saber cuál es la caja con menos clientes, ordenar las cajas de forma ascendente en base a la propiedad `CantidadDeClientesALaEspera`.
    - Utilizar el método `OrderBy` de la biblioteca *LINQ* (`System.Linq`), que tiene como argumento un delegado que recibe un cliente y retorna el valor esa instancia por el que se debe ordenar (en este caso la propiedad `CantidadDeClientesALaEspera`). Ante cualquier duda, consulte la documentación.
  - Una vez ordenada la lista debemos quedarnos con la caja con menos clientes a la espera, es decir, la primera luego de haber ordenado de forma ascendente. Para esto utilizar el método `First` de la biblioteca *LINQ* (`System.Linq`), que retorna el primer elemento de una colección. Ante cualquier duda, consulte la documentación.
  - Para recuperar el cliente de la cola `clientes` utilizar el método `TryDequeue`.
    - Si el método `TryDequeue` retornó un `string` que no sea nulo ni espacios en blanco (puede usar el método estático `string.IsNullOrWhiteSpace`), agregar el cliente a la caja utilizando el método `AgregarCliente` de la caja.
- No recibirá nada y retornará la lista de sub-procesos iniciados por el negocio (`List<Task>`). Esta lista debe incluir los hilos iniciados para la simulación de clientes, la asignación de cajas y los retornados por el método `IniciarAtencion` de las cajas.

Crear una aplicación de consola y en el método `Main`:

1. Instanciar el delegado declarado en `Caja` con una expresión lambda que imprima un mensaje en la consola con el siguiente formato:
   `[Hora actual con formato HH:MM:ss] - Hilo [Id del hilo actual (usar `Task.CurrentId`)] - [Nombre de la caja] - Atendiendo a [nombre del cliente]. Quedan [cantidad de clientes a la espera] clientes en esta caja.`

Ejemplo de resultado esperado: *"13:10:31 - Hilo 1 - Caja 01 - Atendiendo a Christopher Torres. Quedan 2 clientes en esta caja."*

Reemplazar el texto entre corchetes por el dato correspondiente. 2. Instanciar 2 cajas, pasándole al constructor el delegado instanciado en el punto anterior.

3. Instanciar una variable de tipo `List<Caja>` y agregarle las cajas creadas en el punto anterior.
4. Instanciar un `Negocio` y pasarle la lista de cajas del punto anterior.
5. Imprimir por consola el texto: *"Asignando cajas..."*
6. Llamar al método `ComenzarAtencion` de `Negocio`.
7. Utilizar el método estático `WaitAll` de `Task` para que la aplicación no se cierre mientras los hilos retornados por el método `ComenzarAtencion` estén corriendo. Tener en cuenta que `WaitAll` recibe un array como argumento y no una lista (usar el método `ToArray` para convertir la lista en un array).