Compra en Supermercado

NOTA: SI USTED ESTÁ LEYENDO ESTE DOCUMENTO SI HABERLO DESCOMPRIMIDO, CIÉRRELO INMEDIATAMENTE Y DESCOMPRIMA EL CONTENIDO DEL ARCHIVO ZIP QUE RECIBIÓ EN EL ESCRITORIO. UNA VEZ HECHO ESTO, REANUDE SU TRABAJO DESDE ALLÍ. TRABAJAR ANTES DE DESCOMPRIMIR LE HARÁ PERDER SUS CAMBIOS, Y NO TENDRÁ OPORTUNIDAD DE RECLAMAR. LAMENTABLEMENTE ESTO HA OCURRIDO Y NO SE PUEDE HACER NADA.

Para atender a la mayor cantidad de clientes posibles, el supermercado cuenta con N cajas registradoras (numeradas de 0 a $N-1$) que funcionan todas en paralelo. En cada caja registradora los clientes hacen una cola en el mismo orden en van llegando a la caja. Cada cliente tiene una cantidad M_i (que claro está puede ser diferente en cada cliente) de productos a pagar. Considere que todas las cajeras son eficientes por igual y demoran el mismo tiempo en pasar un producto por la caja (con independencia del producto). Por lo tanto si en una caja hay un cliente pagando M_1 productos, y en otra caja hay un cliente pagando $M_2 > M_1$ productos, el cliente de la primera caja terminará antes.

Los clientes siempre se ponen en la cola de la caja que menos personas tenga, aunque la cantidad total de productos en esa cola sea mayor que en otra (puesto que esa información no la puede conocer un cliente sobre los demás clientes que hay en las colas de cada caja).

Si al llegar un cliente hay más de una caja con la misma cantidad de personas en la cola, el cliente escogerá aquella de menor número (en la numeración de 0 a $N-1$). Una vez que un cliente se ha ubicado en una cola de una caja no cambiará de caja aunque la cola de otra caja se vacíe (los clientes son muy obstinados).

A la salida del supermercado hay un solo custodio que verifica la compra de cada cliente. Es dicho custodio quien le pide al conglomerado de cajas por un próximo cliente que haya pagado (ver método `Proximo` más abajo).

Para evitar aglomeraciones en la revisión de salida el custodio pide a las cajas por el próximo cliente a salir y es sólo en ese momento que las cajeras hacen su trabajo para determinar qué cliente sale a continuación. Si dos clientes terminan de pagar a la misma vez, se enviará a la puerta de salida aquel que se encontraba en la caja de menor número (en la numeración de 0 a $N-1$). Una vez que algún cliente llega a la puerta de salida, las cajas paran de trabajar hasta que el custodio vuelva a pedir al próximo cliente. En un momento dado en la punta de la cola de cada caja puede haber clientes que ya han pagado todos sus productos y están a la espera de que los llamen a pasar a la puerta de salida.

Cada vez que se pida `Proximo` se dará una vuelta por todas las cajas que irán pasando un producto a la vez hasta que quede al menos un cliente con todos los productos pasados en al menos una de las cajas. Note que si quedase más de un cliente con todos los productos pasados este saldrá en la siguiente llamada a `Proximo`, y no se cobrará ningún producto a ningún cliente en ninguna caja mientras exista en la punta de alguna otra caja un cliente esperando con todos sus productos ya pagados.

Implementación

Su tarea consiste en implementar un software para el custodio de la puerta. Para eso debe implementar la siguiente `interface`, que maneja el estado del supermercado.

```
public interface ISupermercado
{
    /// <summary>
    /// Lleva un nuevo cliente y se ubica en la caja correspondiente.
    /// </summary>
    void ClienteAPagar(Cliente cliente);

    /// <summary>
    /// El custodio ordena pasar al próximo cliente a la puerta de salida.
    /// </summary>
    bool Proximo();

    /// <summary>
    /// El cliente actualmente en la puerta de salida enseña sus compras.
    /// </summary>
    Cliente EnLaPuerta { get; }

    /// <summary>
    /// Devuelve la cantidad de clientes en espera en la caja correspondiente.
    /// Incluyendo el que puede estar en la punta de la caja con todos los
    /// productos ya pagados.
    /// </summary>
    int ClientesEnCaja(int caja);
}
```

El método `ClienteAPagar` indica la llegada de uno de los clientes del interior del supermercado a la zona de las cajas. Este cliente se ubicará en la caja que menos clientes tenga en cola. El tipo `Cliente` indica además la cantidad de productos que este cliente ha decidido comprar.

```
public class Cliente
{
    public int Productos { get; private set; }

    public Cliente(int productos)
    {
        Productos = productos;
    }
}
```

Si el método `Proximo` devuelve `true` indica que el custodio está listo para recibir al siguiente cliente en la puerta. Cuando se llama a `Proximo` entonces todas las cajas se ponen en funcionamiento hasta que haya al menos un cliente con todos los productos pasados. Puede ser que en más de una caja simultáneamente queden varios clientes con todos los productos pagados. El método devolverá `false` en el caso de que no haya clientes en ninguna caja.

La propiedad `EnLaPuerta` devuelve al cliente que se encuentra actualmente en la puerta resultado de una llamada a `Proximo` que devolvió `true`. En caso de no haber ninguno se debe lanzar una excepción

de tipo `InvalidOperationException`. Por tanto el valor devuelto por esta propiedad solamente cambia tras una siguiente llamada a `Proximo`.

NOTA: Recuerde que se trabaja con referencias, la propiedad `EnLaPuerta` debe devolver por lo tanto **exactamente la misma instancia** de `Cliente` que fue pasada en la correspondiente llamada a `ClienteAPagar` (y de este modo se evaluará). Usted no puede modificar esta clase ni el valor de la propiedad `Productos` (que por demás tiene un `set` privado).

El método `ClientesEnCaja` devuelve la cantidad de clientes en la cola de la caja correspondiente **incluyendo** al cliente que está haciendo su pago.

En la solución entregada usted encontrará una biblioteca de clases con los tipos definidos anteriormente, así como una implementación vacía de la clase `Supermercado` que implementa `ISupermercado`. Es importante que usted trabaje en esta clase, pues su constructor es imprescindible para el correcto funcionamiento del probador.

```
public class Supermercado : ISupermercado
{
    public Supermercado(int cajas)
    {
        throw new NotImplementedException();
    }

    public void ClienteAPagar(Cliente cliente)
    {
        throw new NotImplementedException();
    }

    public bool Proximo()
    {
        throw new NotImplementedException();
    }

    public Cliente EnLaPuerta
    {
        get { throw new NotImplementedException(); }
    }

    public int ClientesEnCaja(int caja)
    {
        throw new NotImplementedException();
    }
}
```

Además, encontrará una aplicación de consola con un conjunto reducido de casos de prueba. Es su responsabilidad adicionar los casos que considere necesarios para validar su implementación.

Ejemplo

En el siguiente ejemplo se cuenta con un supermercado de 2 cajas registradoras. A continuación se describen las acciones que se van realizando y cómo cambia el estado del supermercado.

Acción	EnLaPuerta	Caja 0	Caja 1
`Inicio`	`Nadie`	`Vacía`	`Vacía`
Al comenzar tanto las cajas están vacías y en la puerta no hay nadie (si se preguntara por `EnLaPuerta` *se recibiría una excepción).*			
`ClienteAPagar(8)`	`Nadie`	`A(8)`	`Vacía`
Llega un cliente A, con 8 productos y se ubica en la cola de la caja 0, por ser la de menor número ya que ambas cajas están vacías.			
`ClienteAPagar(5)`	`Nadie`	`A(8)`	`B(5)`
Llega un cliente B, con 5 productos y se ubica en la caja 1 por ser la caja con menos clientes (vacía en este caso).			
`ClienteAPagar(2)`	`Nadie`	`A(8), C(2)`	`B(5)`
Llega un tercer cliente (C, con un 2 producto). Como ambas colas tienen la misma cantidad de clientes se ubica entonces en la de la caja de menor índice (la 0).			
`Proximo`	`B`	`A(3), C(2)`	`Vacía`
El custodio pide `Proximo` *las cajas empiezan a trabajar en paralelo hasta que una queda un cliente en 0 (todos los productos procesados) que es B en este caso. Note que A ha quedado entonces en 3.*			
`ClienteAPagar(3)`	`B`	`A(3), C(2)`	`D(3)`
Llega un nuevo cliente D con 3 productos. Se pone en la cola de la caja 1 porque es la que menos clientes tiene.			
`ClienteAPagar(1)`	`B`	`A(3), C(2)`	`D(3), E(1)`
Llega un nuevo cliente E con 1 producto. Se pone en la cola de la caja 1 porque es la que menos clientes tiene.			
`Proximo`	`A`	`C(2)`	`D(0), E(1)`
El custodio pide `Proximo` *las cajas continúan trabajando. Como la caja 0 y la 1 están con 3 productos a procesar ambas quedan en 0 y por tanto el próximo a salir es A por estar en la caja de menor índice. Note que el cliente al frente en la caja 1 ha quedado en 0.*			
`Proximo`	`D`	`C(2)`	`E(1)`
Una nueva llamada a `Proximo` *como D (al frente en caja 1) estaba en 0 es el que sale antes de que ninguna caja sea procesada.*			
`Proximo`	`E`	`C(1)`	`Vacía`
Sale E que es el que primero queda en 0, note que C ha quedado en 1.			
`Proximo`	`C`	`Vacía`	`Vacía`
`Proximo`	`Excepción`	`Vacía`	`Vacía`
Esta llamada a `Proximo` *devuelve false, y seguirá devolviendo false mientras no se añadan nuevos clientes. Si se pide* `EnLaPuerta` *debe dar excepción.*			

Notas

- Como habrá notado, la lógica de las cajas solamente trabaja cuando hay una llamada a `Proximo`.
- Entre dos llamadas a `Proximo` se pueden realizar cualquier cantidad de llamadas a `ClienteAPagar` sin que esto afecte el funcionamiento del sistema.
- Entre dos llamadas consecutivas a `Proximo` se puede realizar cualquier cantidad de llamadas a `EnLaPuerta`, devolviéndose siempre el mismo objeto `Cliente`.
- Cada objeto `Cliente` creado y añadido con `ClienteAPagar` es diferente, independientemente de que tengan la misma cantidad de productos. Las letras que se le han puesto a los clientes en el ejemplo ha sido solo para facilitar la comprensión del mismo.

- Su implementación debe lanzar `InvalidOperationException` por llamar a `EnLaPuerta` en caso de no haber nadie (`Proximo` dio `false`) o porque no se ha llamado previamente a `Proximo`. Su implementación no necesita realizar ninguna otra validación. Siempre se pasará un valor de parámetro `cliente` diferente de `null`, y con al menos 1 producto, cada vez que se llame a `ClienteAPagar` y siempre se pasará un índice de caja válido cada vez que se llame al método `ClientesEnCaja`.