Cruceros

NOTA: Si usted está leyendo este documento sin haber extraído el compactado que se le entregó, ciérrelo ahora, extraiga todos los archivos en el escritorio, y siga trabajando desde ahí. Es un error común trabajar en la solución dentro del compactado, lo cual provoca que los cambios **no se guarden**. Si usted comete este error y entrega una solución vacía, no tendrá oportunidad de reclamar.

Usted debe implementar el sistema de control de importación de los turistas que llegan en cruceros. El sistema debe determinar si un visitante debe pagar un impuesto según el peso de su equipaje. La aduana tiene establecido para ello un sistema de tarifas según el peso. La tabla a continuación nos muestra un ejemplo:

Peso	Impuesto
Primeros 25 kg	$0
26Kg – 50Kg	$10 / Kg
51Kg – 70Kg	$15 / Kg
71Kg en adelante	$20 / Kg

Una tal tabla se representará con el *array* `{{26, 50, 10}, {51, 70, 15}, {71, int.MaxValue, 20}}`

Si un visitante tiene 60Kg entre **todo** el equipaje que trae, debe pagar $250 ($10 multiplicado por los 25Kg entre 26Kg y 50Kg) más $150 ($15 multiplicado 10Kg entre 51Kg y 60Kg) lo que hace un total de $400.

Por supuesto, la tabla anterior es solamente un ejemplo. En su implementación usted recibirá los valores particulares que se aplicarán a cada peso. Más adelante se precisa el formato y tipo de dato en que recibirá estos valores.

Los turistas llegan en diferentes cruceros y son atendidos a su arribo por la aduana. Teniendo en cuenta las siguientes interfaces:

```
public interface IAduana
{
    void LlegadaDeCrucero(IEnumerable<IVisitante> visitantes);
    IEnumerable<IVisitante> SalidaDeVisitantes();
    IEnumerable<IVisitante> EnColaSinImpuesto(int colaNumero);
    IEnumerable<IVisitante> EnColaConImpuesto(int colaNumero);
    int TiempoMaximoDeEspera { get; }
    int ImpuestoTotalPagado { get; }
}
```

El método `LlegadaDeCrucero` indica la llegada de un nuevo lote de visitantes. Por cada arribo se le informa a la aduana los visitantes de ese crucero (parámetro `visitantes`). Cada visitante se representa por la interface `IVisitante`, mostrada a continuación:

```
public interface IVisitante
{
    string Nombre { get; }
    IEnumerable<IMaleta> Equipaje { get; }
    int CantidadAPagar { get; set; }
}
```

Cada visitante trae varias maletas (propiedad `Equipaje`) y cada maleta tiene una propiedad `Peso` y una propiedad `Descripcion`, según la interface siguiente:

```
public interface IMaleta
{
    string Descripcion { get; }
    int Peso { get; }
}
```

La propiedad `CantidadAPagar` devuelve la cantidad de dinero por concepto de impuestos que tuvo que pagar el visitante. Ud. es el encargado de asignar el valor correcto de esta propiedad según el `Equipaje` del visitante y lo explicado respecto a la aplicación del sistema de tarifas para los impuestos.

La aduana puede atender una cierta cantidad de colas de visitantes que no tienen que pagar impuestos porque el peso total de su equipaje no sobrepasa el peso máximo libre de impuestos y una cantidad de colas para los visitantes que deben pagar impuestos (parámetros `colasSinImpuesto` y `colasConImpuesto` que se le pasan al constructor de aduana respectivamente).

Al llegar un crucero (llamada al método `LlegadaDeCrucero`) la aduana determina el peso del equipaje de cada turista y lo ubica en alguna cola según tenga que pagar impuestos o no. Siempre lo coloca en la **primera** cola del tipo que le corresponde que tenga **menos** cantidad de personas.

El procesamiento de la aduana se simulará con el método `SalidaDeVisitantes` que devuelve un `IEnumerable` con todos los visitantes listos para salir luego de esa llamada. Podrán salir **todos** los visitantes que estén en el **inicio** de cada cola libre de impuestos y todos los visitantes que estando en el **inicio** de las colas para pagar, ya se les haya revisado **todas** sus maletas y pagado el impuesto correspondiente. En cada llamada a `SalidaDeVisitantes` se le disminuirá en 1 la cantidad de maletas a revisar de cada visitante en el inicio de las colas para pagar impuestos (es decir, en la llamada a `SalidaDeVisitantes` un visitante en cola para pagar podrá salir cuando esté en el **inicio** de la cola y **luego** de disminuírsele en **ese** llamado la cantidad de maletas por revisar se llegue a **0**). Los visitantes se devuelven en el siguiente orden: primero los que estaban en colas que no pagan (en el orden de las colas) y luego los que estaban en las colas para pagar (también en el orden de las colas).

Note que cuando un visitante se encuentre en el `IEnumerable<IVisitante>` que devuelve el método `SalidaDeVisitantes` este debe tener ya asignado el valor correcto para la propiedad `CantidadAPagar`.

La propiedad `ImpuestoTotalPagado` indica cuánto ha sido recaudado por concepto de pago de impuestos hasta el momento. Tenga en cuenta que los pasajeros pagan el impuesto justo antes de salir de la cola, por lo que es en ese momento que se debe actualizar dicha propiedad.

La propiedad `TiempoMaximoDeEspera` indica la **cantidad** de llamadas al método `SalidaDeVisitantes` que ocurrieron para que haya podido salir el visitante que **más** se haya demorado en salir hasta el momento en que se consulta dicha propiedad. Note que este no es necesariamente el visitante que más maletas haya tenido porque depende de la cola en que se haya sido ubicado, y la cantidad de personas que tenía

delante. Note que, aunque siempre se ubique a cada visitante en la cola que menos personas tenga esto no garantiza que fluya más rápido que otra con más personas porque depende de la cantidad maletas a revisar por cada una. El tiempo de espera de cada visitante se comienza a medir desde que el visitante entra en la cola correspondiente (es decir, es equivalente a la cantidad de llamadas al método `SalidaDeVisitantes` que ocurren después de la llamada a `LlegadaDeCrucero` que provocó poner al visitante en una cola). Note que para que la propiedad `TiempoMaximoDeEspera` tenga algún valor coherente tiene que haber salido alguien de alguna cola (mientras tanto es 0).

Los métodos `EnColaSinImpuesto(k)` y `EnColaConImpuesto(k)` devuelven respectivamente los visitantes que se encuentran en la cola k-ésima del tipo correspondiente en el momento del llamado.

Usted debe haber recibido una solución de *Visual Studio* con dos proyectos, una biblioteca de clases y una aplicación de consola. En la biblioteca de clases encontrará la clase `Aduana`, que ya implementa la interface `IAduana`. Además, esta clase tiene definido un constructor que recibe todos los parámetros necesarios para definir a la aduana:

```
public class Aduana : IAduana
{
    public Aduana(int libreDeImpuestos, int[,] tablaImpuestos,
                  int colasSinImpuesto, int colasConImpuesto)
    {
    }

    public void LlegadaDeCrucero(IEnumerable<IVisitante> visitantes)
    {
        throw new NotImplementedException();
    }

    public IEnumerable<IVisitante> SalidaDeVisitantes()
    {
        throw new NotImplementedException();
    }

    public IEnumerable<IVisitante> EnColaSinImpuesto(int colaNumero)
    {
        throw new NotImplementedException();
    }

    public IEnumerable<IVisitante> EnColaConImpuesto(int colaNumero)
    {
        throw new NotImplementedException();
    }

    public int TiempoMaximoDeEspera { get; private set; }
    public int ImpuestoTotalPagado { get; private set; }
}
```

En el constructor usted recibirá un primer parámetro `libreDeImpuestos` que representa la cantidad máxima de kilogramos que un visitante puede traer en total en su equipaje sin pagar impuestos (para el ejemplo anterior será el valor 25) y un segundo parámetro `tablaImpuestos` con las tarifas de pago (que para el ejemplo sería el *array* `{{26, 50, 10}, {51, 70, 15}, {71, int.MaxValue, 20}}`)

Se garantiza que `tablaImpuestos` siempre será un *array* bidimensional de tres columnas donde el valor de la primera y segunda columna de una fila indica el intervalo de peso y el valor de la tercera columna indica el impuesto por cada Kg dentro de ese intervalo. Suponga que las cotas de los intervalos se dan correctamente y que los intervalos son consecutivos y ordenados crecientemente.

Los parámetros `colasSinImpuesto` y `colasConImpuesto` indican la cantidad de colas de cada categoría que puede atender la aduana.

NOTA: Todo el código de solución debe estar en este proyecto (biblioteca de clases), pues es el único código que será evaluado. Usted puede adicionar todo el código que considere necesario, pero no puede cambiar los nombres del *namespace*, clases o métodos mostrados. De lo contrario, el probador automático fallará. En particular, es imprescindible que usted no cambie los parámetros del constructor de la clase `Aduana`, ni su orden. Por supuesto, usted puede (y debe) adicionar todo el código que necesite al cuerpo del constructor para inicializar sus estructuras de datos.

Ejemplo

La aplicación de consola se brinda para su conveniencia con algunos ejemplos de prueba. A continuación, se describe un ejemplo (que también aparece en la aplicación de consola) para ilustrar el funcionamiento de la clase `Aduana`.

Inicialmente se crea una aduana con los parámetros correspondientes:

```
int[,] tablaImpuestos = { { 26, 50, 10 }, { 51, 70, 15 }, { 71, int.MaxValue, 20 } };

Aduana aduana = new Aduana(25, tablaImpuestos, 1, 2);
```

Lo que indica que esta aduana atenderá una cola para los que no tengan que pagar y 2 colas para pagar. Una vez creado una aduana se puede indicar la llegada de un crucero:

```
aduana.LlegadaDeCrucero(new IVisitante[]
{
    new Visitante("Juan", new Maleta("Miscelanea", 15), new Maleta("Libros", 5)),
    new Visitante("Pedro", new Maleta("Ropa", 40)),
    new Visitante("Enrique", new Maleta("Alimentos", 5), new Maleta("Electronica", 10)),
    new Visitante("José", new Maleta("Alimentos", 10), new Maleta("Ropa", 15)),
    new Visitante("Miguel", new Maleta("Ropa", 20), new Maleta("TV", 35)),
 });
```

Las clases `Visitante` y `Maleta` se proveen en la aplicación de consola, como una implementación de ejemplo para su uso en los casos de prueba. Usted **no** debe asumir que estas mismas clases se usarán en el probador, por lo que su implementación de `Aduana` **no** puede depender de detalles de implementación de estas clases, sino que debe funcionar con **cualquier** implementación de las interfaces correspondientes.

Usted debe procesar cada uno de estos clientes en el mismo orden en que se pasan al método e irlos ubicando en las respectivas colas:

El primer visitante (`Juan`) tiene un peso total de 20Kg, por lo que no necesita pagar y pasa a la única cola para este propósito que hay. El segundo visitante (`Pedro`) tiene un peso total de 40Kg, por lo que se ubica en la primera de las 2 colas destinadas para pagar. El tercer visitante (`Enrique`) tiene un peso total de 15Kg y se ubica en la cola de los que no pagan. El cuarto visitante (`José`) tiene un peso total de 25Kg, que es **exactamente igual** al peso máximo, por lo que **no** tiene que pagar impuestos, y se ubica también en la cola de los que no pagan. El último cliente (`Miguel`) tiene un peso de 55Kg y se ubica en la segunda cola de los que pagan, que está vacía.

En este punto, el estado de las colas se puede consultar de la siguiente forma:

```
// Cola sin impuestos #1 (no pagan)
foreach (var visitante in aduana.EnColaSinImpuesto(0))
    Console.WriteLine("{0}: {1} maleta(s)", visitante.Nombre, visitante.Equipaje.Count());
// Juan: 2 maleta(s)
// Enrique: 2 maleta(s)
// José: 2 maleta(s)


// Cola con impuesto #1 (si pagan)
foreach (var visitante in aduana.EnColaConImpuesto(0))
    Console.WriteLine("{0}: {1} maleta(s)", visitante.Nombre, visitante.Equipaje.Count());
// Pedro: 1 maleta(s)


// Cola con impuesto #2 (si pagan)
foreach (var visitante in aduana.EnColaConImpuesto(1))
    Console.WriteLine("{0}: {1} maleta(s)", visitante.Nombre, visitante.Equipaje.Count());
// Miguel: 2 maleta(s)
```

En este momento como nadie ha salido los valores de las propiedades `TiempoMaximoDeEspera` y `ImpuestoTotalPagado` son 0

```
Console.WriteLine(aduana.TiempoMaximoDeEspera); //0
Console.WriteLine(aduana.ImpuestoTotalPagado); //0
```

En este punto, con todos los visitantes ubicados, es posible comenzar a realizar la simulación. Para ello se invoca al método `SalidaDeVisitantes` que debe devolver todos aquellos visitantes que salen a la misma vez en el primer paso. Para cada uno de ellos tiene que haberse calculado ya el valor total del impuesto a pagar:

```
// Primer llamado a SalidaDeVisitantes
foreach (var visitante in aduana.SalidaDeVisitantes())
    Console.WriteLine("{0}: ${1}", visitante.Nombre, visitante.CantidadAPagar);
// Juan: $0
// Pedro $150
```

En este caso, salen en el primer paso `Juan` (que tenía 2 maletas, pero como no tiene que pagar solamente demora 1 llamado a `SalidaDeVisitantes`) y `Pedro` (que, aunque tiene que pagar impuestos, como tiene una sola maleta, solo demora un llamado). En el inicio de la cola de los que no pagan queda `Enrique` y en la segunda cola de los que pagan `Miguel`, que ya hizo la revisión de su primera maleta, y por tanto le queda una sola maleta por revisar.

Note que `Pedro` tiene que pagar $150 de sobrepeso, ya que su peso es 40 Kg, por lo que está pasado 15Kg por encima del peso máximo. En las tablas de pesos y costos se establece que entre los 26Kg y los 50Kg se debe pagar $10 por cada Kg extra, luego $10 \cdot (40 - 25) = 150$.

Note como después de este llamado ya las propiedades `TiempoMaximoDeEspera` y `ImpuestoTotalPagado` tienen un valor coherente:

```
Console.WriteLine(aduana.TiempoMaximoDeEspera); //1
Console.WriteLine(aduana.ImpuestoTotalPagado); //150
```

Un segundo llamado a `SalidaDeVisitantes` devuelve:

```
// Segundo llamado a SalidaDeVisitantes
foreach (var visitante in aduana.SalidaDeVisitantes())
    Console.WriteLine("{0}: ${1}", visitante.Nombre, visitante.CantidadAPagar);
// Enrique: $0
// Miguel: $325
```

En este caso la primera cola de los que pagan ya está vacía, así que no devuelve a nadie. En la segunda cola de este tipo, sale `Miguel`, que debe pagar $325, de ellos $250 son por los primeros 25 Kg de sobrepeso (entre 26Kg y 50Kg) y los $75 restantes son por los 5 Kg restantes a $15 cada uno (entre 51Kg y 70Kg).

Ahora ambas colas para pagar impuestos están vacías, pero en la cola de los que no pagan queda `José`.

En este punto llega un nuevo lote de visitantes:

```
aduana.LlegaCrucero(new IVisitante[]
{
    new Visitante("Antonio", new Maleta("Alimentos", 15), new Maleta("PC", 20),
                             new Maleta("Ropa", 25), new Maleta("Ropa", 25)),
    new Visitante("María", new Maleta("Bisutería", 10), new Maleta("Ropa", 40)),
    new Visitante("Alien", new Maleta("Huevos", 40)),
});
```

El sexto visitante (`Antonio`) viene con 85Kg de peso por lo que va para la primera de las colas para pagar. El séptimo visitante (`María`) viene también con sobre peso y pasa a la segunda de las colas para pagar (que está vacía). El octavo visitante (`Alien`) también viene con sobrepeso, por lo que se ubica en la cola de pagar que menos visitantes tenga. Como en este caso ambas tienen la misma cantidad de visitantes, se ubica en la primera cola con declaración.

El estado actual de las colas es el siguiente:

```
// Cola sin impuestos #1 (no pagan)
foreach (var visitante in aduana.EnColaSinImpuesto(0))
    Console.WriteLine("{0}: {1} maleta(s)", visitante.Nombre, visitante.Equipaje.Count());
// José: 2 maleta(s)

// Cola con impuestos #1 (si pagan)
foreach (var visitante in aduana.EnColaConImpuesto(0))
    Console.WriteLine("{0}: {1} maleta(s)", visitante.Nombre, visitante.Equipaje.Count());
// Antonio: 4 maleta(s)
// Alien: 1 maleta(s)
```

```
// Cola con impuestos #2 (si pagan)
foreach (var visitante in aduana.EnColaConImpuesto(1))
    Console.WriteLine("{0}: {1} maleta(s)", visitante.Nombre, visitante.Equipaje.Count());
// María: 2 maleta(s)
```

El siguiente llamado a `SalidaDeVisitantes` devuelve:

```
// Tercer llamado a SalidaDeVisitantes
foreach (var visitante in aduana.SalidaDeVisitantes())
    Console.WriteLine("{0}: ${1}", visitante.Nombre, visitante.CantidadAPagar);
// José: $0
```

Solamente sale `José` que estaba en la cola de los que no pagan impuestos. En la primera cola de los que sí pagan `Antonio` pasa una de sus 4 maletas, y en la segunda, `María` pasa una de sus 2 maletas.

```
// Cuarto llamado a SalidaDeVisitantes
foreach (var visitante in aduana.SalidaDeVisitantes())
    Console.WriteLine("{0}: ${1}", visitante.Nombre, visitante.CantidadAPagar);
// María: $250
```

En el siguiente paso ya sale `María`, y `Antonio` pasa otra de sus 3 maletas.

```
// Quinto llamado a SalidaDeVisitantes
foreach (var visitante in aduana.SalidaDeVisitantes())
    Console.WriteLine("{0}: ${1}", visitante.Nombre, visitante.CantidadAPagar);
// ...
```

En el quinto llamado, solamente la primera cola con declaración tiene visitantes restantes, y como el primero de ellos (`Antonio`) no ha terminado de pasar todas sus maletas (quedaban 2) en este paso no se devuelve a nadie (`IEnumerable` **vacío**, **no** `null`).

```
// Sexto llamado a SalidaDeVisitantes
foreach (var visitante in aduana.SalidaDeVisitantes())
    Console.WriteLine("{0}: ${1}", visitante.Nombre, visitante.CantidadAPagar);
// Antonio: $850
```

En el sexto llamado sale `Antonio`, que debe pagar $10 * (50 - 25) + 15 * (70 - 50) + 20 * (85 - 70) = \$850$

Finalmente, en el último llamado sale el octavo visitante:

```
// Séptimo llamado a SalidaDeVisitantes
foreach (var visitante in aduana.SalidaDeVisitantes())
    Console.WriteLine("{0}: ${1}", visitante.Nombre, visitante.CantidadAPagar);
// Alien: $150
```

A partir de este punto, pueden realizarse cualquier cantidad de llamadas a `SalidaDeVisitantes`, que deben devolver un `IEnumerable` vacío, en tanto no exista otra llamada a `LlegadaDeCrucero` que introduzca nuevos visitantes.

En cualquier momento, las estadísticas generales de la simulación se pueden obtener consultando las propiedades correspondientes.

```
Console.WriteLine(aduana.ImpuestoTotalPagado);    // 1725
Console.WriteLine(aduana.TiempoMaximoDeEspera); // 5
```

El impuesto total pagado es la suma de los valores de `CantidadAPagar` de todos los visitantes que ya han sido devueltos en alguna llamada a `SalidaDeVisitantes`. El tiempo máximo de espera es la cantidad máxima de llamadas a `SalidaDeVisitantes` que fueron necesarias para que cada visitante pasara por la cola correspondiente, a partir del momento en que llegó. En este caso, para `Alien`, que llegó después de la tercera llamada, fueron necesarias cinco llamadas adicionales durante las cuáles estuvo en la cola esperando.

NOTA: Los casos de prueba que aparecen en este proyecto son solamente de ejemplo. Que usted obtenga resultados correctos con estos casos no es garantía de que su solución sea correcta y de buenos resultados con otros ejemplos. De modo que usted debe probar con todos los casos que considere convenientes para comprobar la validez de su implementación.

Usted puede asumir las siguientes condiciones en la solución de su ejercicio:

- Ninguno de los parámetros pasados a ninguno de los métodos de la `interface IAduana` tendrá un valor incorrecto. Es decir, nunca serán `null`, enumerables vacíos, ni índices fuera de rango.
- Usted no necesita implementar la `interface IVisitante` ni la `interface IMaleta`. En la aplicación de consola se le brinda una implementación de cada una para sus pruebas. En la evaluación se podrá usar una implementación diferente, por lo que su código debe funcionar con cualquier implementación de `IVisitante` e `IMaleta`.
- Los valores del *array* que define la tabla de impuestos siempre estarán en orden ascendente por intervalos de peso (que **no necesariamente** es también orden ascendente por costo).
- Si en alguna llamada a `SalidaDeVisitantes` no sale nadie (ya sea porque todos los visitantes restantes aún no han procesado su declaración o porque todas las colas están vacías), usted debe devolver un `IEnumerable` **vacío** pero **NO** un valor `null`.
- Pueden ocurrir llamadas a `LlegadaDeCrucero` sin que todos los pasajeros del crucero anterior hayan salido de la aduana.
- Su implementación de la clase `Aduana` **no** debe modificar en ningún método o propiedad bajo ningún concepto las instancias de `IVisitante` e `IMaleta` que se pasan como argumentos, o sea, los objetos que devuelva en el método `SalidaDeVisitantes` deben ser **exactamente** los mismos que fueron añadidos a través del método `LlegadaDeCrucero`