Sistema de Inventarios

Exámen Extraordinario de Programación II - Curso 2022

NOTA: *Antes de comenzar asegúrese de descompactar el archivo* ***`inventory.zip`*** *y abrir la solución* ***`inventory.sln`*** *en su editor. Asegúrese también de que su código compila, y la aplicación de consola ejecuta (debe lanzar una excepción). Recuerde que todo el código a evaluar debe ir en el archivo* ***`Exam.cs`*** *de la aplicación de consola* ***`exam`****.*

En este ejercicio vamos a implementar un sistema sencillo de inventarios para llevar la contabilidad de un conjunto de productos. De cada producto tendremos un nombre y una cantidad, que podrá ser aumentada o disminuida a medida que cambia el inventario.

Además, cada producto pertenece a una categoría, que a su vez puede pertenecer a otra categoría más general, creándose así un árbol de categorías, donde en cualquier nivel puede haber productos concretos y/o otras subcategorías.

Por ejemplo:

```
- Alimentos:
    - Arroz (10)
    - Carnicos:
        - Pescado (5)
        - Carne de Cerdo (10)
    - Vegetales:
        - Tomate (20)
        - Lechuga (4)
- Electronica:
    - Electrodomesticos:
        - Lavadora (2)
    - Informatica:
        - Ordenador (3)
```

La funcionalidad principal del inventario se encuentra en la interfaz `IInventory`.

```
interface IInventory
{
    // Categoría raíz
    ICategory Root { get; }

    // Navegar por las categorías y productos
    ICategory GetCategory(params string[] categories);
    IProduct GetProduct(string name, params string[] categories);

    // Buscar los productos que cumplen con una condición
    IEnumerable<IProduct> FindAll(Filter<IProduct> filter);
}
```

Como es usual, usted devolverá una instancia de su implementación de esta interfaz en el método estático `Exam.GetInventory` de la clase `Exam` en el archivo `Exam.cs` de la aplicación de consola.

Veremos a continuación cada uno de los métodos que usted debe implementar.

Categorías

Una categoría se define mediante la interfaz `ICategory`. La interfaz `ICategory` se define así (veremos los métodos uno a uno).

```
interface ICategory
{
    string Name { get; }

    // Crear subcategorías
    ICategory CreateSubcategory(string name);

    // Crear o actualizar productos
    void UpdateProduct(string product, int count);

    // Enumerar todas las subcategorías (en este nivel)
    IEnumerable<ICategory> Subcategories { get; }

    // Enumerar todos los productos (en este nivel)
    IEnumerable<IProduct> Products { get; }

    // Categoría padre
    ICategory Parent { get; }
}
```

Todo inventario se crea con una categoría raíz, cuyo nombre es el `string` vacío. Esta es la categoría que se devuelve en la propiedad `Root` de la interfaz `IInventory`.

Para obtener una categoría arbitraria, se puede usar el método `GetCategory` de la interfaz `IInventory` que recibe un array de `string` con los nombres de las categorías intermedias.

Por ejemplo, para obtener la categoría `"Informática"` se invocaría a este método de la siguiente forma:

```
IInventory inv = Exam.GetInventory();
ICategory informatica = inv.GetCategory("Electronica", "Informatica");
```

En caso de no existir la categoría pedida usted debe lanzar una excepción de tipo `ArgumentException`.

Una vez que tenemos una referencia a una categoría, es posible utilizarla para crear nuevas subcategorías. Por ejemplo para crear una subcategoría `Moviles`

dentro de la categoría `Informatica`:

```
ICategory moviles = informatica.CreateSubcategory("Moviles");
```

Por supuesto, una vez que esta categoría ha sido creada, desde el inventario original es posible obtener exactamente la misma referencia:

```
Debug.Assert(moviles == inv.GetCategory("Electronica",
                                        "Informatica",
                                        "Moviles"));
```

La propiedad `Parent` en `ICategory` devuelve una referencia a la categoría padre. En el caso de ser la categoría raíz, devuelve una referencia a sí misma (nunca `null`).

Productos

En cualquier categoría, el método `UpdateProduct` aumenta (o disminuye) la cantidad de cualquier producto.

Si un producto existe, su cantidad se modifica en el valor `count`. Si un producto no existe en esta categoría, se crea automáticamente cuando se invoque este método con la cantidad pasada.

```
// Disminuye en 1 la cantidad de ordenadores
informatica.UpdateProduct("Ordenador", -1);

// Crea un nuevo producto
moviles.UpdateProduct("Samsung Galaxy", 10);
```

Por supuesto ningún producto puede bajar de `0` ni crearse con una cantidad negativa. En cualquiera de estos casos usted debe lanzar una excepción de tipo `ArgumentException`.

La propiedad `Subcategories` enumera todas las subcategorías que son hijas immediatas de esta categoría.

La propiedad `Products` enumera todos los productos **con cantidad mayor que cero** immediatamente en esta categoría. Esta propiedad devuelve instancias de la interfaz `IProduct` que simplemente almacena el nombre y cantidad, así como una referencia a la categoría a la que pertenece:

```
interface IProduct
{
    string Name { get; }
    int Count { get; }
    ICategory Parent { get; }
}
```

En la interfaz `IInventory` el método `GetProduct`, muy similar a `GetCategory`,

devuelve directamente el producto correspondiente, dado su nombre y las categorías a las que pertenece. Por ejemplo:

```
IProduct tomate = inv.GetProduct("Tomate", "Alimentos", "Vegetales");
Debug.Assert(tomate.Count == 20);
```

Filtrado de productos

El método `FindAll` de la interfaz `IInventory` enumera todos los productos que cumplen con cierta condición, definida por el delegado `Filter`:

```
delegate bool Filter<T>(T item);
```

Por ejemplo, para encontrar todos los productos que tienen menos de 3 elementos:

```
foreach(var product in inv.FindAll(p => p.Count < 5))
{
    // Verificando que efectivamente tiene menos de 5
    Debug.Assert(product.Count > 0 && product.Count < 5);
}
```

Ejemplos de prueba

En la aplicación de consola encontrará un ejemplo de prueba muy similar a lo que hemos visto hasta ahora, que le permitirá verificar que los métodos básicos funcionan.

NOTA: El ejemplo de prueba es insuficiente para garantizar que su código está 100% correcto. En particular, los métodos de iteradores no se verifican. Es su responsabilidad adicionar tantos casos de prueba como considere necesario para garantizar la correctitud de su solución.

¡Éxitos a todos!