🛒 Kata: Lista de la compra

Queremos construir una clase para gestionar una lista de la compra a través de una única función.

La lista comienza vacía. El usuario puede añadir productos, eliminarlos o vaciar la lista completamente. Todas las operaciones se realizan mediante instrucciones en forma de texto.

Tu tarea es implementar una clase que interprete estas instrucciones y mantenga el estado de la lista.

🧱 Reglas generales

- Solo puede haber **una clase pública**, con **un único método público** que reciba una instrucción (`string`) y devuelva el estado actual de la lista.
- El método debe devolver un `string` con los productos actuales de la lista, separados por comas.
- Los productos deben aparecer ordenados alfabéticamente (ignorando mayúsculas/minúsculas).
- Los nombres de producto **no distinguen mayúsculas**: `"Pan"` y `"pan"` son el mismo producto.

✅ Acciones que debe soportar

Añadir productos

- Instrucción: `añadir <nombre> [cantidad]`
- Si no se indica cantidad, se asume `1`.
- Si el producto ya existe en la lista, se suma la nueva cantidad a la anterior.
- Ejemplos:
    - `añadir pan` → `"pan x1"`
    - `añadir Pan 2` → `"pan x3"`

Eliminar productos

- Instrucción: `eliminar <nombre>`
- Elimina completamente el producto de la lista.
- Si el producto no existe, el método debe devolver exactamente:

```
Unset
El producto seleccionado no existe
```

Vaciar la lista

- Instrucción: `vaciar`
- Elimina todos los productos de la lista.

📤 Formato de salida

Después de cada instrucción válida, se debe devolver la lista completa como un `string`, con los productos separados por comas.

Cada producto debe mostrarse en el siguiente formato:

```
Unset
<nombre> x<cantidad>
```

Ejemplo:

```
Unset
"leche x2, pan x3"
```

Si la lista está vacía, se devuelve una cadena vacía: `""`.

🔁 Ejemplo de flujo

```
Unset
"añadir pan"              // "pan x1"
"añadir leche 2"          // "leche x2, pan x1"
"añadir Pan 2"            // "leche x2, pan x3"
"eliminar arroz"          // "El producto seleccionado no
existe"
"eliminar pan"            // "leche x2"
"vaciar"                  // ""
```

📋 Criterios de evaluación y buenas prácticas

Además de que tu solución funcione correctamente, se valorarán los siguientes aspectos:

✅ Buena cobertura de tests y aplicación de TDD

✅ Clean Code y buen naming

✅ Buen uso de commits y ciclo de trabajo

- Cada commit debe representar un paso del ciclo TDD:
    - Un commit para cada **test que pasa (verde)**.
    - Un commit para cada **refactor (si lo hay)**.
- Usa el siguiente formato para los mensajes de commit:

```
Unset
[verde] - Descripción clara del test que pasa

[refactor] - Descripción clara del cambio estructural o de
estilo
```