AddProductMainJava

1. Los objetos creados no se están usando correctamente

```
Supplier supplier = new Supplier(nextNumber, supplierName, "N/A", "N/A");
AnimalCategory cat = new AnimalCategory(nextNumber, category, animal + " " + size);
```

2. El ID usa el número del supplier, no el del producto

```
int nextNumber = getNextIdNumber(addedProducts, idPrefix);
String id = idPrefix + String.format("%02d", nextNumber);
Supplier supplier = new Supplier(nextNumber, supplierName, "N/A", "N/A");
```

3. El JSON se sobreescribe siempre

```
new FileWriter("products.json")
```

4. El inventario de Store nunca se guarda

```
store.getInventory().addProduct(product)
```

5. Si el usuario elige 2 o 3 en error, igual vuelve a pedir datos

```
if (!Arrays.asList(VALID_CATEGORIES).contains(category)) {
    option = handleError(sc);
    continue;  // <-- esto sigue el ciclo aunque escoja "Salir"
}
```

POSIBLES SOLUCIONES :

1. Usar los objetos que estás creando:

Agregar `Supplier` al objeto `Product` (modificando la clase Product para que lo reciba)

Guardar suppliers en una lista `List<Supplier>` si no quieres modificar Produc**t**

2. Que el ID del producto no dependa del número del Supplier

Usar un contador independiente solo para productos

Obtener el número desde el JSON ya guardado para que no se repita

3. Evitar que JSON se sobrescriba

Leer el JSON antes de agregar nuevos productos

4. Guardar el inventario del Store, no solo la lista temporal

Guardar `store.getInventory().getProducts()` en JSON

Eliminar `addedProducts` y usar solo el inventario del store

Sincronizar ambas listas siempre

5. Que el programa sí salga cuando el usuario lo elige

Cambiar `continue` por `break`

```
if (option != 1) break;
```

Main.Java

1. Cada producto agregado en opción 1 no se guarda en el inventario del store

```
case 1 -> {
    AddProductMain.main(null);
}
```

Solución:
Modificar `AddProductMain` para recibir el inventario:

```
case 1 -> AddProductMain.addProduct(store.getInventory(), sc);
```

2. ID de Customer y Employee siempre es 1

```
Customer cust = new Customer(1, cname, address, cphone);
Employee emp = new Employee(1, ename, role);
```

Error: todos tendrán el mismo ID.

Solución: hacerlo autoincremental:

```
Customer cust = new Customer(UUID.randomUUID().hashCode(), cname,
address, cphone);
Employee emp = new Employee(UUID.randomUUID().hashCode(), ename,
role);
```

3. Solo permite agregar 1 producto a la orden
Después de agregar 1 producto, no permite añadir más.

Solución (extra): envolverlo en un ciclo:

```
String addMore;
do {
    // lógica de agregar producto
    System.out.print("Add another product? (y/n): ");
    addMore = sc.nextLine();
} while (addMore.equalsIgnoreCase("y"));
```

4. No valida si hay inventario suficiente

Cuando se compra un producto no verifica stock.

Solución:

```
if (selected.getStock() >= qty) {
    selected.setStock(selected.getStock() - qty);
    currentOrder.addDetail(new OrderDetail(selected, qty));
    System.out.println("Added to order");
} else {
    System.out.println("Not enough stock available");
}
```

5. Order ID y Bill ID siempre son 1

```
currentOrder = new Order(1, cust, emp);
Bill inv = new Bill(1, currentOrder);
```

Solución: igual que en Customer, usar ID dinámico:

```
currentOrder = new Order(UUID.randomUUID().hashCode(), cust, emp);
```

Bill inv = new Bill(UUID.randomUUID().hashCode(), currentOrder);

6. El mensaje “No order created yet” tiene un símbolo raro

System.out.println(" No order created yet.");

Puede generar errores visuales en consola.

Cambiar a:

System.out.println("No order created yet.");

v0.3.0 MainJava

1. Clase LoginSystem no está importada o no existe

Error probable:

cannot find symbol class LoginSystem

Solución:

- **Crea la clase si no existe, o impórtala si está en otro paquete:**

import ec.edu.espe.petshop.controller.LoginSystem;

2. InvalidProductDataException está en model, pero parece un util

No es error crítico, pero es mala práctica que una excepción de validación esté en model.

Recomendación:

Mover a:

ec.edu.espe.petshop.utils.InvalidProductDataException

3. List<?> res = pc.search(q); limita el uso

No se rompe, pero pierdes tipo específico y no puedes manipular productos.

Mejor:

```
List<Product> res = pc.search(q);
```