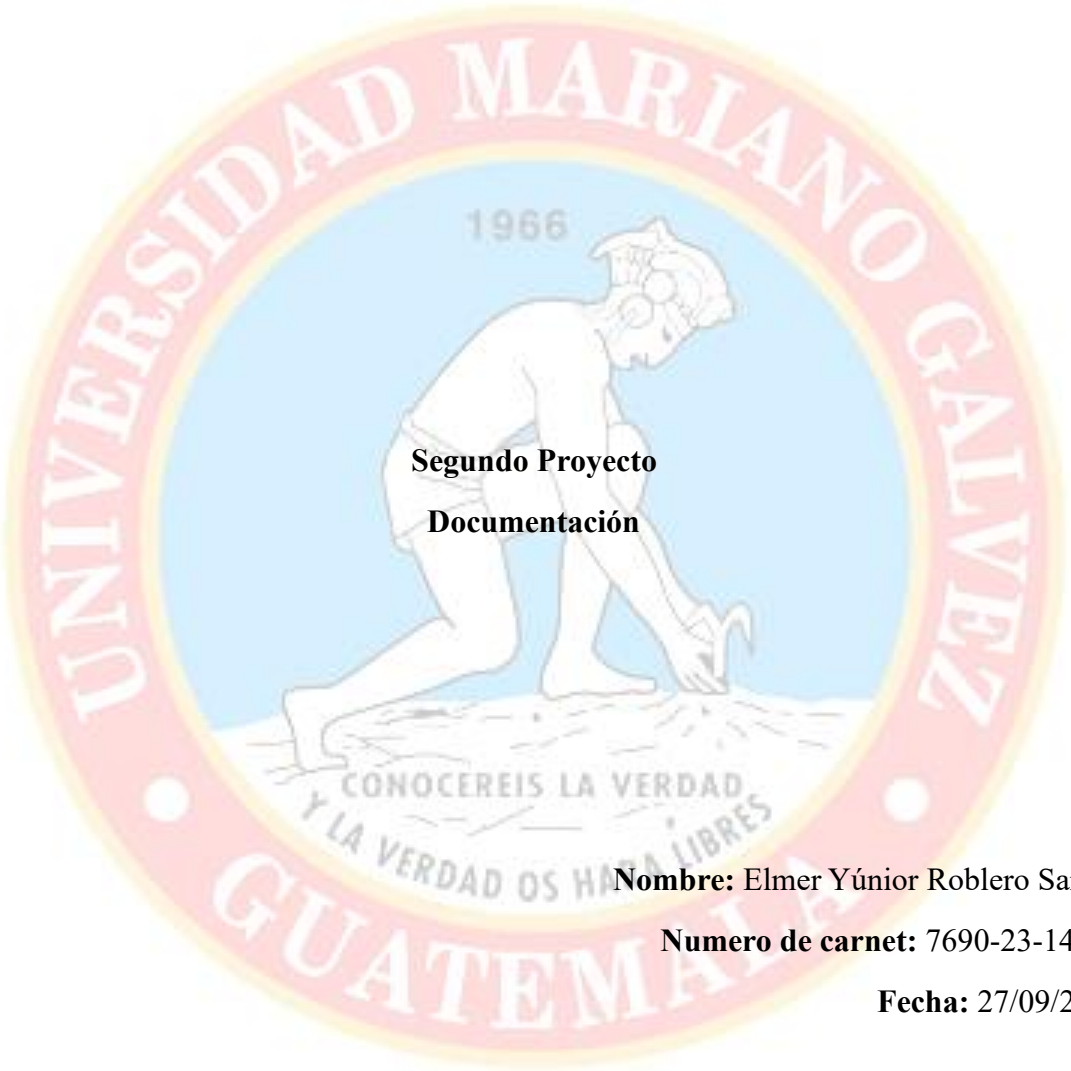
Universidad Mariano Gálvez

Centro Regional: Boca del Monte

Facultad: Ingeniería en sistemas

Curso: Programación II

Sección: A



Segundo Proyecto

Documentación

Nombre: Elmer Yúnior Roblero Santos

Numero de carnet: 7690-23-14847

Fecha: 27/09/2024

Guatemala, de 2024

Documentación del Sistema de Gestión de Productos

1. ProductState (Interfaz)

Descripción:

Define la interfaz para los diferentes estados que puede tener un producto.

Métodos:

- void handle(Product product): Maneja las operaciones específicas del estado sobre el producto. Cada estado concreto debe implementar este método.

---

2. AvailableState (Clase Concreta)

Descripción:

Implementa el estado "Disponible" de un producto.

Métodos:

- void handle(Product product): Muestra que el producto está disponible.

---

3. SoldOutState (Clase Concreta)

Descripción:

Implementa el estado "Agotado" de un producto.

Métodos:

- void handle(Product product): Muestra que el producto está agotado.

---

4. Product (Clase Base)

Descripción:

Representa un producto genérico. Esta clase contiene atributos como el nombre y precio del producto, así como su estado actual.

Atributos:

- String name: Nombre del producto.
- double price: Precio del producto.
- ProductState state: El estado actual del producto.

Métodos:

- String getName(): Retorna el nombre del producto.
- double getPrice(): Retorna el precio del producto.
- void setState(ProductState state): Establece el nuevo estado del producto.
- void checkState(): Llama al método handle() del estado actual para gestionar el estado del producto.

---

5. ConcreteProduct (Clase Abstracta)

Descripción:

Clase abstracta que extiende a Product y que servirá como base para productos específicos (ej. electrónicos, ropa).

---

6. Electronics (Clase Concreta)

Descripción:

Una subclase de ConcreteProduct que representa un producto del tipo "Electrónica".

Constructor:

- Electronics(String name, double price): Inicializa el producto con un nombre y precio.

---

7. Clothing (Clase Concreta)

Descripción:

Una subclase de ConcreteProduct que representa un producto del tipo "Ropa".

Constructor:

- Clothing(String name, double price): Inicializa el producto con un nombre y precio.

---

8. ProductFactory (Clase)

Descripción:

Implementa el patrón Factory para crear productos concretos según su tipo.

Métodos:

- static Product createProduct(String type, String name, double price): Crea una instancia de Product según el tipo proporcionado (electronics, clothing). Si el tipo no es soportado, lanza una excepción IllegalArgumentException.

---

9. CatalogLoader (Clase)

Descripción:

Carga productos desde un archivo JSON y los convierte en instancias de Product usando la fábrica de productos.

Métodos:

- List<Product> loadCatalog(String filePath): Lee un archivo JSON ubicado en filePath y carga la lista de productos en memoria. Retorna una lista de objetos Product.

---

10. FileStorage (Clase)

Descripción:

Guarda productos en un archivo JSON.

Métodos:

- void saveToFile(List<Product> products, String filePath): Guarda la lista de productos en un archivo JSON en la ubicación especificada por filePath.

---

11. Main (Clase)

Descripción:

La clase principal para ejecutar el programa, cargar productos, verificar su estado y guardarlos en un archivo JSON.

Flujo:

1. Carga los productos desde el archivo catalogo.json.
2. Verifica el estado de cada producto.
3. Guarda los productos en productos_guardados.json.
4. Simula un cambio de estado de "Disponible" a "Agotado" para el primer producto de la lista.

JSON de Entrada y Salida

Archivo de Entrada (catalogo.json):

Es un archivo JSON que contiene los productos a cargar en el sistema.

Formato:

json

Copiar código

```
[
    { "type": "electronics", "name": "Laptop", "price": 999.99 },
    { "type": "clothing", "name": "T-shirt", "price": 19.99 }
]
```

Archivo de Salida (productos_guardados.json):

Un archivo JSON que contiene los productos que han sido cargados y procesados.

Formato:

json

Copiar código

```
[
    { "name": "Laptop", "price": 999.99, "type": "electronics" },
    { "name": "T-shirt", "price": 19.99, "type": "clothing" }
]
```

Patrones de Diseño Utilizados

1. **Patrón State**:
    - Define el comportamiento de los productos basado en su estado. Los estados concretos (AvailableState, SoldOutState) implementan la interfaz ProductState.
    - Facilita la extensión futura al agregar más estados sin modificar el código del producto.
2. **Patrón Factory**:
    - Permite la creación de objetos de tipo Product sin especificar la clase exacta a instanciar.
    - Mejora la mantenibilidad del código al centralizar la lógica de creación de objetos.

Uso de Librería JSON

La implementación utiliza la librería org.json para manipular datos JSON. Los métodos CatalogLoader.loadCatalog() y FileStorage.saveToFile() hacen uso de esta librería para cargar y guardar productos en archivos JSON.

Dependencias:

Asegúrate de agregar la librería org.json a tu proyecto. Si estás usando Maven, puedes incluir la siguiente dependencia:

xml

Copiar código

```
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20240303</version>
</dependency>
```

Flujo General del Sistema

1. **Carga del Catálogo**:
    - El sistema carga productos desde un archivo JSON.
    - Usa el patrón Factory para crear productos concretos (Electronics, Clothing).
2. **Verificación del Estado**:
    - Cada producto inicia en el estado "Disponible".
    - El estado del producto se puede verificar usando el método checkState().
3. **Cambio de Estado**:
    - Se puede cambiar el estado del producto dinámicamente usando el método setState().
4. **Guardado en Archivo**:
    - Después de procesar los productos, estos se guardan en un archivo JSON.