Sistema de Gestión Tech Vault

Escuela: Universidad Autónoma de Guadalajara

Por: Lucas Patrick Castineiras y Carlo Virgilio

Registro: 509489 y 15064883

Maestra: JESSICA FERNANDA ROSAS AGRAZ

Fecha: 4/12/2025

Materia: Programacion Orientada a Objetos

Planteamiento del Escenario

El proyecto nace de la necesidad de una tienda emergente de tecnología, **"Tech Vault"**, especializada en hardware de alto rendimiento. Actualmente, la tienda enfrenta problemas operativos críticos debido a la gestión manual de su inventario, lo que ocasiona errores en precios y desconocimiento del stock real.

Se requiere una solución digital centralizada que permita gestionar un inventario heterogéneo (Laptops, Monitores y Accesorios), procesar órdenes de compra mediante un "carrito virtual" y generar reportes financieros, asegurando que la información no se pierda al cerrar el sistema.

Objetivos del Proyecto

El objetivo principal es desarrollar una solución de software robusta utilizando el paradigma de **Programación Orientada a Objetos (POO)** en Python que cumpla con:

1. **Gestión de Inventario Heterogéneo:** Administrar productos con características únicas (ej. RAM para Laptops vs. Resolución para Monitores) mediante Herencia.
2. **Procesamiento de Ventas:** Implementar un sistema de carrito de compras capaz de agrupar múltiples productos y calcular totales.
3. **Persistencia de Datos:** Utilizar archivos JSON para guardar y cargar el estado del inventario automáticamente.
4. **Seguridad y Reportes:** Restringir el acceso a reportes de ventas sensibles mediante autenticación y validar entradas para evitar errores (precios negativos o tipos de datos incorrectos).

Marco Teórico y Tecnologías

- **Lenguaje:** Python 3.x.
- **Encapsulamiento:** Protección de atributos sensibles (como el precio) para evitar modificaciones inválidas mediante *setters* y *getters*.
- **Herencia:** Creación de una jerarquía de clases donde Laptop, Accessory y Monitor heredan comportamientos de una clase base Product.
- **Polimorfismo:** Capacidad de diferentes objetos de responder al mismo método (to_dict) de formas distintas según su estructura interna.
- **Composición:** La clase Order no hereda de productos, sino que *contiene* una lista de productos, estableciendo una relación "tiene-un".

Arquitectura y Explicación del Código

El sistema se dividió en tres módulos principales para asegurar la modularidad y el mantenimiento del código.

Modelo de Clases (models.py)

Este archivo contiene la lógica de negocio y las definiciones de los objetos.

- **Clase Product (Padre):** Define los atributos comunes (name, sku) y encapsula el precio (_price). Se implementa validación en el método set_price para lanzar una excepción ValueError si se intenta ingresar un precio negativo.
- **Clases Hijas (Laptop, Accessory, Monitor):** Heredan de Product. Utilizan super().__init__ para inicializar los datos base y añaden sus atributos específicos (RAM, Compatibilidad, Tamaño/Resolución).
    - **Polimorfismo en to_dict():** Cada clase implementa su propia versión de este método para serializar sus datos específicos a un formato diccionario compatible con JSON.
- **Clase Order:** Implementa el patrón de **Composición**. Tiene una lista self.products = []. Sus métodos permiten agregar objetos Product y calcular el total iterando sobre la lista y sumando los precios mediante get_price().

Persistencia de Datos (data_manager.py)

Este módulo se encarga de la entrada y salida de datos (I/O) utilizando la librería estándar json.

- **Clase DataManager:**
    - **save_inventory:** Convierte la lista de objetos en una lista de diccionarios (usando el método polimórfico to_dict()) y la escribe en inventory.json. Maneja excepciones de tipo IOError.
    - **load_inventory:** Lee el archivo JSON. Lo más destacado es su lógica de **reconstrucción de objetos**: al leer los datos, verifica el campo "type" (Laptop, Accessory, Monitor) e instancia la clase correcta antes de agregarla a la lista en memoria. Esto asegura que los métodos específicos de cada clase sigan funcionando al recargar el programa.

Lógica Principal e Interfaz (main.py)

Es el punto de entrada (Entry Point) de la aplicación.

- **Ciclo de Vida:** Utiliza un bucle while True para mantener el menú interactivo activo hasta que el usuario decida "Guardar y Salir".
- **Manejo de Excepciones:** Todo el bloque de entrada de datos está envuelto en una estructura try-except para capturar errores como ValueError (si el usuario ingresa letras en lugar de números para el precio o RAM), evitando que el programa colapse.
- **Diccionarios para Reportes:** Se utiliza un diccionario sales_totals para acumular las ventas por categoría en tiempo real.
- **Seguridad:** La opción de reportes (Opción 5) solicita una contraseña (ADMIN_PASSWORD = "admin123") antes de mostrar la información financiera.

Resultados y Funcionamiento

Al ejecutar el sistema, se obtuvieron los siguientes resultados funcionales:

1. **Inicio:** El sistema carga exitosamente datos previos si existe el archivo inventory.json.
2. **Alta de Productos:** El usuario puede agregar Laptops, Accesorios y Monitores. El sistema valida correctamente que no se ingresen precios negativos.
3. **Ventas:** El carrito de compras permite seleccionar productos por su índice. Al finalizar, muestra un recibo detallado y actualiza las estadísticas de ventas internas.
4. **Cierre:** Al seleccionar la opción 6, el sistema serializa todos los objetos en memoria y actualiza el archivo JSON, garantizando la persistencia.

Ejemplo de estructura JSON generada:

JSON


```
[
    {
        "type": "Laptop",
        "name": "Alienware",
        "sku": "DELL-01",
        "price": 2500.0,
        "ram": 32
    },
    {
        "type": "Monitor",
        "name": "Samsung Odyssey",
        "sku": "SAM-99",
        "price": 450.0,
        "size": 27.0,
        "resolution": "4K"
    }
]
```


6. Conclusiones

La implementación de este proyecto demostró la efectividad de la Programación Orientada a Objetos para resolver problemas de gestión de datos complejos.

- **Escalabilidad:** Gracias a la herencia, agregar la nueva categoría Monitor fue sencillo y no requirió reescribir la lógica base de precios o inventario, simplemente se creó una nueva clase y se actualizó el cargador de datos.
- **Robustez:** El uso de encapsulamiento y manejo de excepciones asegura que errores humanos (como typos en inputs numéricos) no detengan la operación de la tienda.

- **Persistencia:** La integración con JSON permite que el software sea útil en un entorno real, manteniendo la información segura entre sesiones.