UNIVERSIDAD TECNOLÓGICA DEL URUGUAY
ITR SUROESTE · FRAY BENTOS
INGENIERÍA MECATRÓNICA
UC de Programación IV · 2022/2
Profesor: Giovani Bolzan Cogo

Práctico de laboratorio II - Programación Orientada a Objetos	Periodo: de 19/09/2025 hasta 03/10/2025

Integrantes

Felipe Morrudo – 5.582.493-4
Hector Pereira – 5.582.582-5
Santiago Moizo – 5.165.430-5
Aldrin Pacheco –

Resumen

Este trabajo implementa un sistema de gestión de clientes, vendedores, productos y compras usando Programación Orientada a Objetos en Python. El diseño se documenta con un diagrama de clases UML donde se modelan las relaciones clave: composición entre Compra e ItemCompra, y asociación/agregación con Producto, además de los vínculos con Cliente y Vendedor. La persistencia se resuelve con archivos de texto en formato JSON Lines (un registro por línea), con escritura atómica para evitar corrupción y funciones de carga que reconstituyen instancias a partir de índices por código. Se desarrolló un menú de consola para crear, listar y borrar entidades y para registrar compras con varios ítems, calculando totales y almacenando metadatos de factura. Los resultados muestran que el guardado/carga es consistente y que el modelo es extensible para incorporar validaciones adicionales, descuentos e impuestos.

Índice

1. Introducción

El presente informe documenta el desarrollo de un sistema de gestión de clientes, vendedores, productos y compras, implementado en Python bajo el paradigma de Programación Orientada a Objetos (POO). El trabajo corresponde al Práctico de Laboratorio II y aplica de forma integrada modelado con UML, diseño de clases y persistencia en archivos de texto.

El proyecto se apoya en un diagrama UML que describe las entidades principales y sus relaciones (composición entre Compra e ItemCompra, y asociaciones con Cliente, Vendedor y Producto). Este modelo guió la implementación, definiendo responsabilidades, encapsulamiento y navegación entre objetos.

Para la persistencia se utilizó el formato JSON Lines (JSONL), que almacena cada registro en una línea independiente y facilita la escritura y lectura incremental. Se desarrollaron funciones genéricas para serializar y rehidratar instancias, manteniendo consistencia entre modelo y datos en disco.

El informe presenta los objetivos, la fundamentación teórica, la metodología de desarrollo, los resultados (incluyendo UML final y evidencias de ejecución) y una discusión de conclusiones y posibles mejoras.

2. Objetivos

Este práctico tiene como objetivo general el modelado Orientado a Objetos (O.O.) de un sistema de gestión de ventas de una empresa ficticia, aplicando análisis de estructuras de datos y modelado en diagrama UML (Unified Modeling Language).

De los objetivos puntuales, destácanse los siguientes.

- Modelar el dominio con clases y relaciones UML consistentes con el código.
- Implementar persistencia en archivos TXT usando JSONL con escritura atómica.
- Proveer un menú de consola para crear, consultar y borrar entidades y registrar compras.
- Rehidratar objetos desde disco con funciones `load_*` e índices por código para resolver referencias.

3. Fundamentación teórica

El desarrollo del código se realiza en Python, un lenguaje de sintaxis clara y modelo de objetos uniforme. Su tipado dinámico con anotaciones opcionales facilita documentar expectativas y expresar reglas de negocio de forma directa, apoyándose en una amplia biblioteca estándar para manejo de archivos, datos y ejecución sin agregar complejidad innecesaria.

3.1. Fechas y tiempos

dateutil.parser aporta un analizador flexible de fechas capaz de reconocer múltiples formatos habituales sin configuraciones extensas.

dateparser complementa lo anterior al interpretar expresiones en lenguaje natural (“hoy”, “mañana”, “15/03/25”), reduciendo errores de carga y normalizando la información temporal en objetos datetime comparables y ordenables.

3.2. Datos en archivos de texto

Con ***json*** se guardan y leen listas, diccionarios, números y textos en un formato estándar. Esto facilita almacenar el estado del sistema, recargarlo posteriormente o intercambiar datos con otras herramientas.

3.3. Rutas y archivos

pathlib.Path permite trabajar con rutas como objetos (no como simples cadenas). Así podemos preguntar si un archivo existe, crear carpetas o leer y escribir texto con métodos directos, y el código funciona igual en distintos sistemas operativos.

3.4. Ejecución del programa

sys da acceso a cosas del entorno de ejecución. Por ejemplo, ***sys.argv*** trae los argumentos con los que se abrió el programa. Esto permite cambiar el comportamiento según esos parámetros, sin tocar la lógica principal.

3.5. Aclaración de datos en función

Aunque Python no exige declarar tipos, con ***typing*** podemos aclarar qué datos recibe o devuelve una función (por ejemplo, ***Iterable*** indica que acepta listas, tuplas u otros objetos recorribles). Estas anotaciones facilitan la comprensión y permiten que el editor detecte errores antes de la ejecución.

Las librerías empleadas permiten manejar ***fechas***, rutas y archivos con ***pathlib.Path***, ajustar la ejecución mediante argumentos con ***sys*** (por ejemplo, ***sys.argv***) y escribir código más claro indicando los datos esperados en cada parte.

Para la persistencia se utiliza el formato ***JSON Lines*** (un objeto por línea, codificación UTF-8) en la carpeta `./db`, con los archivos: `dbClientes.txt`, `dbVendedores.txt`, `dbProductos.txt` y `dbCompras.txt`.

1. **Lectura robusta de datos.** La función `_read_jsonl(path)` abre cada archivo y retorna una lista de registros. Soporta dos variantes: (a) *JSON Lines* (procesa línea a línea con `json.loads`); (b) *JSON array completo* (si comienza con "[" lo decodifica de una vez). Si el archivo no existe o está vacío, devuelve una lista vacía, evitando errores al iniciar.

2. **Escritura atómica y segura.** `_write_jsonl_atomic(path, rows)` evita archivos corruptos: crea el directorio si falta, escribe primero en un temporal (`.tmp`) y lo reemplaza de forma atómica al finalizar. Cada registro se serializa en una línea JSON, preservando acentos y caracteres (`ensure_ascii=False`).

3. **Guardado centralizado.** `save_all` sincroniza las listas con sus TXT (`clientes`, `vendedores`, `productos` y `compras`). Para las tres primeras usa `to_dict`; para compras delega en `to_record`, que almacena códigos de cliente, vendedor y productos, cantidades y metadatos de factura (`nFactura`, `modoPago`, `fechaVencFactura`, `valido`, `total`, `listado`). Así se minimiza duplicación y se mantiene integridad.

4. **Carga y reconstrucción.** `load_clientes`, `load_vendedores` y `load_productos` leen sus TXT y reconstruyen instancias con `from_dict`. Para **compras**, `load_compras` lee cada registro, reconstruye fechas en ISO, resuelve referencias de cliente, vendedor y producto contra índices previos, arma `ItemCompra` y recompone la `Compra`. Si falta alguna referencia, informa el problema y omite solo lo necesario, manteniendo consistencia.

Este esquema permite iniciar el sistema aunque no haya archivos, escribir sin riesgo de cortes a mitad de guardado y reconstruir relaciones entre entidades a partir de códigos, manteniendo consistencia y evitando duplicaciones.

3.6. Programación Orientada a Objetos (POO)

La POO propone modelar el dominio con *clases* y crear *objetos* que colaboran entre sí. Sus cuatro pilares mantienen el modelo ordenado: el **encapsulamiento** protege el estado mediante interfaces claras; la **abstracción** selecciona los detalles relevantes para el problema; la **herencia** permite especializar sin duplicar (p. ej., `Cliente` y `Vendedor` comparten `Persona`); y el **polimorfismo** posibilita que objetos distintos respondan a la misma operación. En conjunto, reducen acoplamiento, mejoran cohesión y facilitan el mantenimiento del código [Mor00, RJB99].

3.7. Relaciones UML: asociación, agregación, composición y multiplicidad

En UML no todas las flechas significan lo mismo [RJB99, Fow04, UML]. La **asociación** es un vínculo simple: dos clases se conocen pero pueden existir por separado (p. ej., `Compra` con `Cliente`/`Vendedor`). La **agregación** describe una relación todo–parte *débil*, donde las partes no dependen del ciclo de vida del todo (el `listado` en `Compra` apunta a `Producto`, que existe por sí mismo). La **composición** es más fuerte: si el todo muere, las partes también (los `ItemCompra` sólo existen dentro de `Compra`). Las **multiplicidades** (1, 0.,1, 1..∗, etc.) completan el modelo indicando cuántas instancias se permiten en cada extremo. Todo esto se refleja en el diagrama final.

3.8. Persistencia y serialización

El sistema utiliza *JSON Lines* (JSONL), un objeto por línea, lo que facilita la inspección manual, las operaciones incrementales y evita depender de una base de datos. Cada clase implementa `to_dict`/`to_record` y `from_dict`/`from_record` para traducirse a diccionarios y reconstruirse. En `Compra` se persisten *referencias* (códigos) a `Cliente`, `Vendedor` y `Producto`, evitando duplicación y manteniendo la fuente de verdad en las entidades maestras [ECM17, Pyt].

3.9. Integridad referencial y escritura atómica

La carga desde disco valida referencias mediante diccionarios índice (`clientes_by`, `vendedores_by`, `productos_by`). Si un código no existe, se emite una advertencia sin afectar el resto. La escritura se realiza de forma *atómica*: primero se genera un archivo temporal y sólo al final se reemplaza el definitivo, garantizando que, ante fallos, el original permanezca intacto.

3.10. Normalización y manejo de fechas

Las fechas se normalizan a **ISO 8601** (`YYYY-MM-DD`) para evitar ambigüedades. Se aceptan entradas flexibles (`DD/MM/AAAA` o textos comunes) con `dateutil`/`dateparser`, mientras que la clase `Fecha` encapsula parseo, conversión (`to_iso`, `toString`) y validación básica. De este modo, internamente todo es consistente y externamente la experiencia resulta sencilla [Pyt].

3.11. Estructuras de datos y eficiencia

Los "índices" por código (`dict`) permiten accesos promedio $O(1)$ a clientes, vendedores y productos. Esto agiliza la reconstrucción de compras al resolver referencias sin recorrer listas. Con pocos datos la mejora es marginal, pero con cientos o miles la diferencia es significativa.

3.12. Tipado y validación

Las *type hints* documentan contratos y ayudan al editor/CI a detectar errores temprano. En la entrada de usuario aplicamos `try`/`except` y conversiones explícitas (`int`, `float`). Los cargadores desde disco comprueban estructura y claves obligatorias: si algo viene mal, lo ignoramos de forma controlada y seguimos [Lut13, Gut16].

3.13. Contratos, estado y consistencia

`ItemCompra` asegura cantidades positivas. `Compra` mantiene un estado de validez y un total: si no está efectivizada, lo calcula al vuelo desde los ítems; si está validada, conserva el total persistido para trazabilidad. Estas reglas hacen el sistema predecible y con efectos colaterales acotados.

4. Metodología

4.1. Herramientas y material

Con *Draw.io* se replicó el diagrama UML base de la práctica, incluyendo las clases `Fecha`, `Producto`, `Compra`, `Persona`, `Vendedor` y `Cliente`, junto con sus atributos y métodos.

Para la implementación se utilizó *Python* 3.10 en la plataforma *Google Colab*, que permite edición colaborativa en línea al almacenar el código en la nube y habilitar el acceso simultáneo de varios usuarios.

4.2. Procedimiento

1. **Diagrama UML.** Se analizo el diagrama UML inicial proporcionado en la consigna de la práctica, con el fin de pensar en posibles modificaciones a implementar para facilitar la gestion del sistema. A continuación, en la Fig.1 se observa el diagrama UML inicial.

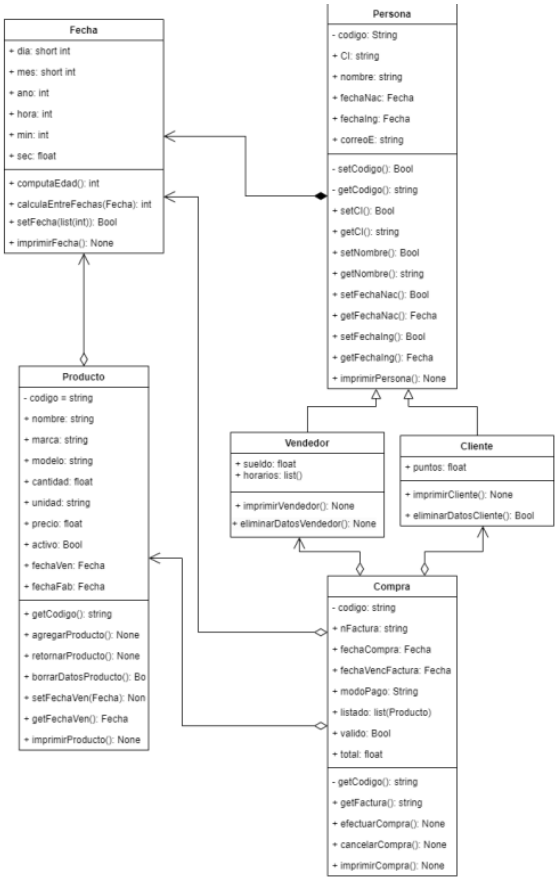

Figura 1: Diagrama UML inicial

Para optimizar el sistema se implementó la clase *ItemCompra*, cuya función es encapsular los datos necesarios para calcular el subtotal de una línea de factura. Cada línea almacena el producto, su cantidad y precio unitario, y calcula el monto total multiplicando estos valores. La clase *Compra* obtiene el total sumando todos los *ItemCompra*.

A su vez, se incorporó la clase *Menu*, encargada de la lógica de interacción con el usuario: gestiona las operaciones de creación, modificación y eliminación sobre las demás clases del sistema, centralizando la interfaz y la recolección de datos.

2. **Código en Python.** Finalizado el diagrama UML, se implementó el código en *Python* aplicando los conceptos de P.O.O. Se incorporaron atributos, métodos, getters y setters necesarios para el funcionamiento de las clases, además de librerías que simplifican su manejo.

En la clase *Menu* se desarrolló la lógica de navegación que permite al usuario interactuar con el sistema, ofreciendo las siguientes opciones:

- Consultar información
- Crear información
- Borrar información
- Menú compras
- Salir

3. **Almacenamiento de datos.** Las listas en memoria se sincronizan con archivos ".txt" en formato JSON Lines en `./db` (`dbProductos.txt`, `dbClientes.txt`, `dbVendedores.txt`, `dbCompras.txt`). Al iniciar, si el archivo existe se carga como JSONL o arreglo JSON; si no, se parte de listas vacías. El guardado es atómico (`.tmp` → reemplazo) para evitar corrupciones.

   Las compras se persisten de forma referencial (códigos de cliente, vendedor y productos, cantidades y metadatos) y al cargar se reconstruyen y validan dichas referencias. Véase el ítem "Funciones de lectura y guardado (TXT/JSON Lines)" para más detalle.

4. **Verificación y validación.** Se probaron unidades (`Fecha`, `Cliente`, `Vendedor`, `Producto`, `ItemCompra` y cálculos de subtotal/total) y luego el flujo integrado con un conjunto mínimo (2 clientes, 2 vendedores, 3 productos y 2 compras), usando `save_all("./db", ...)` y posterior recarga. Criterios de aceptación:

   - `dbClientes.txt`, `dbVendedores.txt`, `dbProductos.txt` y `dbCompras.txt` se generan sin errores.
   - Las compras se guardan por referencias (códigos y cantidades) y, al recargar, cada código se resuelve a un objeto válido.
   - Los totales previos al guardado coinciden con los recalculados tras la carga.
   - Si se fuerza un código inexistente, el sistema lo informa al reconstruir, evitando datos corruptos.

   Estas pruebas confirmaron que las clases funcionan de forma aislada y, en conjunto con la persistencia, mantienen coherencia con los requisitos.

4.3. Justificación y Alcance de la Metodología

El enfoque metodológico se basó en cuatro pilares: diseño, implementación, persistencia y validación. Se utilizó **Google Colab** como entorno de desarrollo por su soporte colaborativo en la nube y **Python 3.10** por sus capacidades de programación orientada a objetos. El almacenamiento se resolvió con **JSON Lines (JSONL)** en archivos `.txt`, facilitando la serialización y deserialización de objetos de forma legible y sin dependencias externas.

4.4. Criterios de Validación

La validación del sistema se basó en distintos criterios para garantizar confiabilidad y coherencia. Se verificó la consistencia de datos en operaciones de creación, guardado y carga de clientes, vendedores, productos y compras, evitando pérdida de información. También se comprobó la validez de las relaciones, asegurando que cada `ItemCompra` refiera a un `Producto` existente. Otro criterio fue el manejo de errores en las entradas, de modo que datos inválidos fueran rechazados sin comprometer la base. Finalmente, se revisaron manualmente reportes como `imprimirCliente` e `imprimirCompra`, confirmando que la salida coincidiera con los datos ingresados.

4.5. Limitaciones

La metodología presenta algunas limitaciones consideradas como oportunidades de mejora. En primer lugar, no se implementaron pruebas unitarias automatizadas, por lo que la validación fue manual y con menor alcance. Además, la persistencia en archivos JSONL resulta adecuada para un entorno académico y de pequeña escala, pero no es escalable para grandes volúmenes de datos o accesos concurrentes. Por último, el manejo de errores se limita a validaciones básicas de entrada, sin cubrir casos más complejos como concurrencia o corrupción de archivos.

4.6. Conexión con los Objetivos

Finalmente, la metodología descrita asegura cumplir con los objetivos planteados al garantizar:

- Un modelo UML completo y consistente.
- Una implementación funcional en Python siguiendo principios de POO.
- Persistencia de datos simple y confiable mediante archivos JSONL.
- Validación del correcto funcionamiento a través de casos de prueba controlados.

5. Resultados

5.1. Nuevo UML

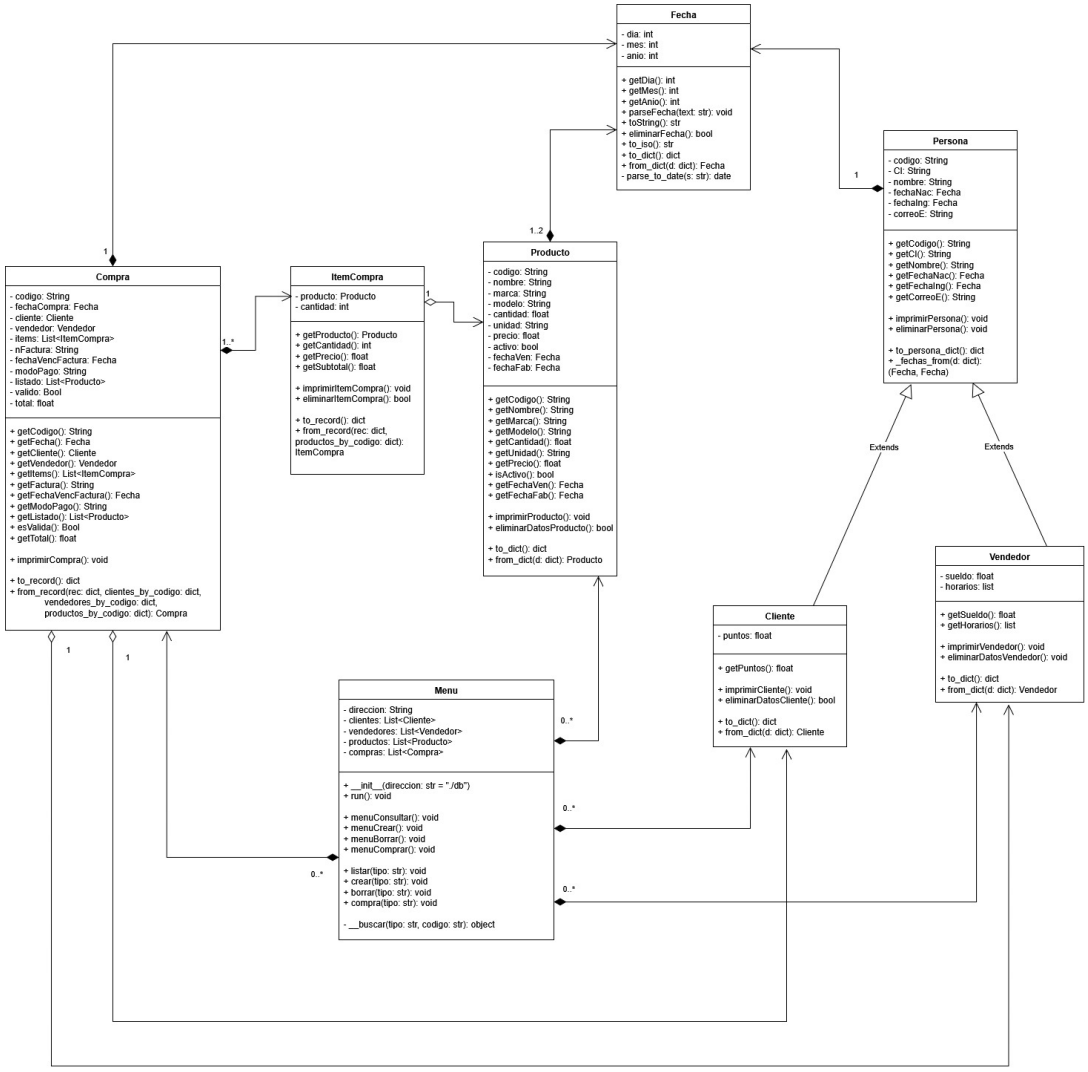

Figura 2: Diagrama UML final. Fuente: Elaboración própia

Cambios respecto al UML inicial. En el diagrama final, `Persona` quedó como clase base, de la cual heredan `Cliente` y `Vendedor`, evitando duplicar atributos como código, CI, fechas y correo. `Producto` se amplió con marca, modelo, unidad, estado (activo) y fechas de fabricación y vencimiento, describiendo mejor el inventario.

`Compra` pasó a ser el núcleo: además de fecha, cliente, vendedor e ítems, incorpora número de factura, modo de pago, un listado explícito de productos, estado y total. Para ello, `ItemCompra` actúa como vínculo entre compra y producto, con cantidad y subtotal.

Finalmente, se añadió `Menu` como interfaz principal, responsable de orquestar las operaciones y conectar con la persistencia en archivos. En conjunto, el UML final refleja con mayor fidelidad la implementación y permite futuras extensiones sin afectar la estructura existente.

5.2. Clases

El sistema se estructuró siguiendo los principios de la Programación Orientada a Objetos (POO), organizando la lógica en distintas clases que modelan tanto las entidades del dominio como las operaciones del sistema. De forma resumida:

- **Fecha**: encapsula día, mes y año, empleado en nacimientos, ingresos y compras.
- **Persona**: clase base con atributos comunes a `Cliente` y `Vendedor`.
- **Cliente** y **Vendedor**: subclases de `Persona` con atributos propios, como puntos acumulados o sueldo y horarios.
- **Producto**: describe los artículos a vender, con identificación, stock y fechas de fabricación y vencimiento.
- **ItemCompra**: vincula un producto con una cantidad, formando parte de una compra.
- **Compra**: centraliza la transacción, integrando cliente, vendedor, productos, fecha, factura y total.
- **Menu**: gestiona la interacción con el usuario y el flujo de operaciones, incluida la compra.

5.3. Guardado y lectura de archivos

La persistencia se implementó con archivos de texto en formato `JSON Lines` (JSONL), donde cada objeto se almacena en una línea independiente. Esto facilita la lectura secuencial, la compatibilidad con distintos lenguajes y la inspección manual.

Para el **guardado**, la función `save_all` recorre las listas de `Cliente`, `Vendedor`, `Producto` y `Compra`, serializando cada objeto con `to_dict` o `to_record`. Luego, `_write_jsonl_atomic` escribe en un archivo temporal (`.tmp`) y lo reemplaza de forma atómica, evitando corrupciones en caso de error.

En la **lectura**, `_read_jsonl` interpreta cada línea como JSON y devuelve una lista de diccionarios. A partir de ellos, las funciones `load_clientes`, `load_vendedores`, `load_productos` y `load_compras` reconstruyen las instancias mediante `from_dict` o `from_record`. En `Compra`, además, se valida la integridad referencial verificando que cada ítem apunte a un `Producto`, `Cliente` y `Vendedor` cargados previamente.

Este mecanismo asegura:

- Integridad de datos al manipular listas de objetos entre sesiones.
- Independencia del sistema frente a gestores de bases de datos externos.
- Sencillez en la depuración y validación de archivos.

En la Figura 3 se muestra un ejemplo de registro almacenado en el archivo `dbCompras.txt`, donde se observa la referencia a códigos de cliente, vendedor y productos.

```
{"cliente_codigo": "", "codigo": "C01", "fecha": "1990-01-01", "fechaVencFactura": "
2024-04-20", "items": [{"cantidad": 2, "producto_codigo": ""}, {"cantidad": 10, "
producto_codigo": "P01"}, {"cantidad": 20, "producto_codigo": "P02"}], "listado": ["", "
P01", "P02"], "modoPago": "Tarjeta", "nFactura": "F001", "total": 11009.2, "valido":
true, "vendedor_codigo": ""}
```


Figura 3: Ejemplo de registro en formato JSONL para una compra.

5.4. Resumen de validaciones

Criterio	Estado	Evidencia
Archivos `db*.txt` generados sin errores	**OK**	Logs de ejecución y archivos en `./db`
Compras por referencias y reconstrucción correcta	**OK**	Carga con `load_compras` (IDs válidos)
Totales antes/después del guardado coinciden	**OK**	Comparación de `getTotal()` vs recálculo
Referencias inválidas se reportan sin romper datos	**OK**	Mensajes de advertencia y omisión controlada

Tabla 1: Criterios de validación y resultados.

6. Conclusiones

El proyecto logró implementar un modelo de dominio coherente en POO con clases bien delimitadas (`Persona`, `Cliente`, `Vendedor`, `Producto`, `ItemCompra`, `Compra` y `Menu`), reflejado en un UML final consistente con las decisiones de diseño. La separación entre lógica de negocio y persistencia permitió serializar y rehidratar objetos de forma confiable usando JSON Lines, con escritura atómica para evitar corrupción y diccionarios–índice para preservar la integridad referencial y mejorar el rendimiento en accesos y cargas. Desde el punto de vista técnico, el sistema demostró que con un conjunto acotado de patrones (encapsulamiento, composición de `Compra`–`ItemCompra`, asociaciones con `Cliente`/`Vendedor` y normalización de fechas) es posible obtener un flujo completo: alta, listado, borrado, compra y posterior relectura desde disco, manteniendo coherencia de datos y un contrato claro de métodos públicos. La experiencia de uso mejoró gracias a validaciones simples de entrada y a una impresión estandarizada de entidades, lo que facilitó la verificación manual. También quedaron visibles límites razonables: la ausencia de pruebas unitarias automatizadas, el uso de archivos de texto como única capa de persistencia y la falta de control de concurrencia. Estas restricciones no afectan el objetivo del laboratorio, pero marcan un camino de evolución: migrar a una base de datos ligera, incorporar tests para casos borde (referencias inexistentes, formatos de fecha irregulares, cantidades inválidas), y agregar reglas de negocio como descuentos, impuestos y manejo de stock con entradas/salidas. En síntesis, se cumplió el objetivo de modelar, implementar y persistir el dominio propuesto con una arquitectura simple, extensible y didáctica. El diseño actual deja el terreno preparado para crecer sin romper interfaces: la capa de datos es intercambiable, las entidades están encapsuladas y el UML captura las relaciones esenciales que guían futuras ampliaciones.

Referencias

[Dow12] A. Downey. *Think Python: How to Think Like a Computer Scientist.* Green Tea Press, 2012.

[ECM17] ECMA International. The json data interchange syntax, standard ecma-404, 2017.

[Fow04] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language.* Addison-Wesley, 3rd edition, 2004.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[Gut16] J. V. Guttag. *Introduction to Computation and Programming Using Python.* MIT Press, 2nd edition, 2016.

[Lut13] M. Lutz. *Learning Python.* O'Reilly Media, 5th edition, 2013.

[Mor00] F. Morero. *Introducción a la OOP.* Grupo EIDOS, Montevideo, 2000.

[Pyt] Python Software Foundation. Python 3 documentation.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1999.

[UML] UML-diagrams.org. Uml class and object diagrams overview.

A. Recursos

- **Carpeta general (Drive):** https://drive.google.com
- **Diagrama UML:** https://drive.google.com
- **Código (Google Colab):** https://colab.research.google.com
- **Documento (Google Docs):** https://docs.google.com
- **Overleaf / coordinación:** https://www.overleaf.com/

B. CÓDIGO-FUENTE EN LENGUAJE PYTHON

```
from dateutil import parser
import dateparser
import sys
import json
from pathlib import Path
from typing import Iterable

# Guardado en base de datos


_DB_FILES = {
    "productos":  "dbProductos.txt",
    "vendedores": "dbVendedores.txt",
    "clientes":   "dbClientes.txt",
    "compras":    "dbCompras.txt",
}

def _read_jsonl(path: Path) -> list[dict]:
    '''Lee un archivo JSONL o un JSON array y devuelve una lista de dicts.

    Soporta dos formatos:
      1) **JSON Lines (JSONL)**: un objeto JSON por linea.
      2) **JSON Array**: el archivo completo contiene un array JSON.

    Si el archivo no existe o esta vacio, devuelve `[]`.

    Args:
        path (Path): Ruta del archivo a leer.

    Returns:
        list[dict]: Lista de registros decodificados.

    Raises:
        json.JSONDecodeError: Si el contenido no es JSON valido.

    Ejemplo:
        >>> filas = _read_jsonl(Path("./db/dbClientes.txt"))
        >>> isinstance(filas, list)
        True
    '''
    if not path.exists():
        return []
    txt = path.read_text(encoding="utf-8").strip()
    if not txt:
        return []
    if txt.lstrip().startswith("["):
```

```
        return json.loads(txt)
    out = []
    for line in txt.splitlines():
        line = line.strip()
        if line:
            out.append(json.loads(line))
    return out

def _write_jsonl_atomic(path: Path, rows: Iterable[dict]) -> None:
    '''Escribe registros como JSONL de forma atomica y con UTF-8.

    Crea el directorio si no existe. Primero escribe a un archivo temporal
    (`.tmp`) y luego reemplaza el archivo destino, evitando archivos corruptos
    ante fallos a mitad de escritura.

    Args:
        path (Path): Ruta del archivo destino.
        rows (Iterable[dict]): Registros a serializar (uno por linea).

    Returns:
        None

    Raises:
        OSError: Si hay fallos de E/S al escribir o renombrar.
        TypeError: Si algun elemento de `rows` no es serializable a JSON.

    Ejemplo:
        >>> _write_jsonl_atomic(Path("./db/dbProductos.txt"),
        ...                     (p.to_dict() for p in listaProductos))
    '''
    path.parent.mkdir(parents=True, exist_ok=True)
    tmp = path.with_suffix(path.suffix + ".tmp")
    with tmp.open("w", encoding="utf-8") as f:
        for r in rows:
            f.write(json.dumps(r, ensure_ascii=False, sort_keys=True) + "\n")
    tmp.replace(path)

def save_all(
    base_dir: str | Path,
    listaClientes,
    listaVendedores,
    listaProductos,
    listaCompras
):
    '''Persiste todas las listas de objetos del sistema en archivos TXT (JSONL).

    Escribe:
      - clientes   -> dbClientes.txt
      - vendedores -> dbVendedores.txt
      - productos  -> dbProductos.txt
      - compras    -> dbCompras.txt

    Para clientes, vendedores y productos se usa `obj.to_dict()`.
    Para compras se usa `compra.to_record()` (formato relacional por codigos).

    Args:
        base_dir (str | Path): Carpeta base que contiene los TXT.
        listaClientes: Lista de instancias `Cliente`.
        listaVendedores: Lista de instancias `Vendedor`.
        listaProductos: Lista de instancias `Producto`.
        listaCompras: Lista de instancias `Compra`.
```

```
    Returns:
        None

    Side effects:
        Crea/actualiza archivos dentro de `base_dir`.

    Ejemplo:
        >>> save_all("./db", listaClientes, listaVendedores, listaProductos, listaCompras)
    '''
    base = Path(base_dir)
    _write_jsonl_atomic(base / _DB_FILES["clientes"],   (c.to_dict() for c in listaClientes)
    )
    _write_jsonl_atomic(base / _DB_FILES["vendedores"], (v.to_dict() for v in
    listaVendedores))
    _write_jsonl_atomic(base / _DB_FILES["productos"],  (p.to_dict() for p in listaProductos
    ))
    _write_jsonl_atomic(base / _DB_FILES["compras"],    (c.to_record() for c in listaCompras
    ))

def load_clientes(path: Path) -> list[Cliente]:
    '''Carga clientes desde un TXT (JSONL/JSON) y devuelve instancias `Cliente`.

    Usa `Cliente.from_dict(d)` para reconstruir cada objeto. Las lineas que
    no puedan decodificarse o mapearse se informan y se omiten.

    Args:
        path (Path): Ruta al archivo de clientes (p.ej. ./db/dbClientes.txt).

    Returns:
        list[Cliente]: Lista de clientes validos.

    Ejemplo:
        >>> clientes = load_clientes(Path("./db/dbClientes.txt"))
        >>> len(clientes) >= 0
        True
    '''
    clientes = []
    for d in _read_jsonl(path):
        try:
            clientes.append(Cliente.from_dict(d))
        except Exception as e:
            print(f"Warning: could not load cliente {d}: {e}")
    return clientes

def load_vendedores(path: Path) -> list[Vendedor]:
    '''Carga vendedores desde un TXT (JSONL/JSON) y devuelve instancias `Vendedor`.

    Usa `Vendedor.from_dict(d)` para reconstruir cada objeto. Registros con
    errores se informan y omiten.

    Args:
        path (Path): Ruta al archivo de vendedores (p.ej. ./db/dbVendedores.txt).

    Returns:
        list[Vendedor]: Lista de vendedores validos.

    Ejemplo:
        >>> vendedores = load_vendedores(Path("./db/dbVendedores.txt"))
        >>> all(hasattr(v, "getSueldo") for v in vendedores)
        True
```

```
    '''
    vendedores = []
    for d in _read_jsonl(path):
        try:
            vendedores.append(Vendedor.from_dict(d))
        except Exception as e:
            print(f"Warning: could not load vendedor {d}: {e}")
    return vendedores

def load_productos(path: Path) -> list[Producto]:
    '''Carga productos desde un TXT (JSONL/JSON) y devuelve instancias `Producto`.

    Usa `Producto.from_dict(d)` para reconstruir cada objeto. Registros con
    errores se informan y omiten.

    Args:
        path (Path): Ruta al archivo de productos (p.ej. ./db/dbProductos.txt).

    Returns:
        list[Producto]: Lista de productos validos.

    Ejemplo:
        >>> productos = load_productos(Path("./db/dbProductos.txt"))
        >>> any(p.getPrecio() > 0 for p in productos)
        True
    '''
    productos = []
    for d in _read_jsonl(path):
        try:
            productos.append(Producto.from_dict(d))
        except Exception as e:
            print(f"Warning: could not load producto {d}: {e}")
    return productos

def load_compras(path: Path, clientes_by_codigo, vendedores_by_codigo, productos_by_codigo):
    '''Carga compras desde TXT (JSONL) y devuelve instancias `Compra`.

    Reconstruye cada compra a partir de un registro relacional:
      - fecha en ISO (yyyy-mm-dd) -> `Fecha(d, m, y)`
      - `cliente_codigo` y `vendedor_codigo` se resuelven con los *indices*
        `clientes_by_codigo` y `vendedores_by_codigo`.
      - `items` se reconstruyen con `ItemCompra(producto, cantidad)`, donde
        el `producto` se busca en `productos_by_codigo`.
      - Campos opcionales de factura: `nFactura`, `modoPago`,
        `fechaVencFactura` (ISO), `valido`, `total`, `listado`.

    Los registros con referencias inexistentes (cliente/vendedor/producto)
    se informan y se omiten parcialmente o totalmente, segun el caso.

    Args:
        path (Path): Ruta al archivo de compras (p.ej. ./db/dbCompras.txt).
        clientes_by_codigo (dict[str, Cliente]): indice por codigo de cliente.
        vendedores_by_codigo (dict[str, Vendedor]): indice por codigo de vendedor.
        productos_by_codigo (dict[str, Producto]): indice por codigo de producto.

    Returns:
        list[Compra]: Lista de compras reconstruidas.

    Ejemplo:
        >>> clientes = load_clientes(Path("./db/dbClientes.txt"))
        >>> vendedores = load_vendedores(Path("./db/dbVendedores.txt"))
```

```
    >>> productos = load_productos(Path("./db/dbProductos.txt"))
    >>> compras = load_compras(Path("./db/dbCompras.txt"),
    ...                        {c.getCodigo(): c for c in clientes},
    ...                        {v.getCodigo(): v for v in vendedores},
    ...                        {p.getCodigo(): p for p in productos})
    >>> isinstance(compras, list)
    True
'''
compras = []
if not path.exists():
    return compras

with path.open("r", encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        rec = json.loads(line)

        # Fecha de compra
        y, m, d = map(int, rec["fecha"].split("-"))
        fecha = Fecha(d, m, y)

        # Fecha vencimiento de factura (si existe)
        fecha_venc = None
        if rec.get("fechaVencFactura"):
            y2, m2, d2 = map(int, rec["fechaVencFactura"].split("-"))
            fecha_venc = Fecha(d2, m2, y2)

        cliente = clientes_by_codigo.get(rec["cliente_codigo"])
        vendedor = vendedores_by_codigo.get(rec["vendedor_codigo"])

        if cliente is None or vendedor is None:
            print(f"Warning: Cliente/Vendedor missing in {rec['codigo']}, skipping")
            continue

        items = []
        for i in rec["items"]:
            producto = productos_by_codigo.get(i["producto_codigo"])
            if producto is None:
                print(f"Warning: Producto {i['producto_codigo']} not found, skipping")
                continue
            items.append(ItemCompra(producto, i["cantidad"]))

        compra = Compra(
            codigo=rec["codigo"],
            fechaCompra=fecha,
            cliente=cliente,
            vendedor=vendedor,
            items=items,
            nFactura=rec.get("nFactura", ""),
            modoPago=rec.get("modoPago", ""),
            fechaVencFactura=fecha_venc,
            valido=bool(rec.get("valido", False)),
            total=float(rec.get("total", 0.0)),
            listado=[productos_by_codigo.get(p) for p in rec.get("listado", []) if p in
productos_by_codigo],
        )
        compras.append(compra)

return compras
```

```
# Menu

class Menu:
  def __init__(self, direccion: str):
    # Para usarse luego
    self.__direccion = direccion

    # Leer listas de la base de datos
    self.__clientes   = load_clientes(Path(self.__direccion + "/dbClientes.txt"))
    self.__vendedores = load_vendedores(Path(self.__direccion + "/dbVendedores.txt"))
    self.__productos  = load_productos(Path(self.__direccion + "/dbProductos.txt"))

    # Listas de solo de codigos para cargar compras
    clientes_by   = {c.getCodigo(): c for c in self.__clientes}
    vendedores_by = {v.getCodigo(): v for v in self.__vendedores}
    productos_by  = {p.getCodigo(): p for p in self.__productos}

    # Leer lista de compras de la base de datos
    self.__compras = load_compras(Path(self.__direccion + "/dbCompras.txt"), clientes_by,
    vendedores_by, productos_by)

  def run(self): # ------------------------------------------------------------> Run
    # Menu principal
    print("\n")
    print("////////// Menu principal ///////////")
    print("1 - Consultar informacion")
    print("2 - Crear informacion")
    print("3 - Borrar informacion")
    print("4 - Menu compras")
    print("0 - Salir")

    print("\n")
    switch = input("Ingrese una opcion:")
    if switch == "1":   self.menuConsultar()
    elif switch == "2": self.menuCrear()
    elif switch == "3": self.menuBorrar()
    elif switch == "4": self.menuComprar()
    elif switch == "0":
      print("... Guardando datos y saliendo del programa ...")
      save_all(
        self.__direccion,
        self.__clientes,
        self.__vendedores,
        self.__productos,
        self.__compras
      )
      sys.exit()
    else:
      print("--- Opcion invalida ---")
      self.run()


  def menuConsultar(self): # ------------------------------------------------> Menu
    Consultar
    print("\n")
    print("////////// Consultar ///////////")
    print("1 - Listar clientes")
    print("2 - Listar vendedores")
    print("3 - Listar productos")
    print("4 - Listar compras")
```

```
    print("0 - Volver")

    print("\n")
    switch = input("Ingrese una opcion:")
    if switch =="1":   self.listar("clientes")
    elif switch =="2": self.listar("vendedores")
    elif switch =="3": self.listar("productos")
    elif switch =="4": self.listar("compras")
    elif switch =="0": self.run()
    else:
      print("--- Opcion invalida ---")
      self.menuConsultar()
    self.run()


def menuCrear(self): # ----------------------------------------------------> Menu Crear
    print("\n")
    print("////////// Menu crear //////////")
    print("1 - Crear cliente")
    print("2 - Crear vendedor")
    print("3 - Crear producto")
    print("0 - Volver")

    print("\n")
    switch = input("Ingrese una opcion:")
    if switch =="1":   self.crear("clientes")
    elif switch =="2": self.crear("vendedores")
    elif switch =="3": self.crear("productos")
    elif switch =="0": self.run()
    else:
      print("--- Opcion invalida ---")
      self.menuCrear()


def menuBorrar(self): # ----------------------------------------------------> Menu Borrar
    print("\n")
    print("////////// Menu borrar //////////")
    print("1 - Borrar cliente")
    print("2 - Borrar vendedor")
    print("3 - Borrar producto")
    print("0 - Volver")

    print("\n")
    switch = input("Ingrese una opcion:")
    if switch =="1":   self.borrar("clientes")
    elif switch =="2": self.borrar("vendedores")
    elif switch =="3": self.borrar("productos")
    elif switch =="0": self.run()
    else:
      print("--- Opcion invalida ---")
      self.menuBorrar()


def menuComprar(self): # ----------------------------------------------------> Menu
    Comprar
    print("\n")
    print("////////// Menu comprar //////////")
    print("1 - Crear compra")
    print("2 - Suspender compra")
    print("0 - Volver")
```

```
  print("\n")
  switch = input("Ingrese una opcion:")
  if switch =="1":   self.compra("crear")
  elif switch =="2": self.compra("suspender")
  elif switch =="0": self.run()
  else:
    print("--- Opcion invalida ---")
    self.menuComprar()


def listar(self, tipo): # ----------------------------------------------------> Listar
  print("\n")
  if tipo =="clientes":
    print("======== Listado de clientes ========")
    for c in self.__clientes:
        c.imprimirCliente()

  elif  tipo =="vendedores":
    print("======== Listado de vendedores ========")
    for v in self.__vendedores:
        v.imprimirVendedor()

  elif  tipo =="productos":
    print("======== Listado de productos ========")
    for p in self.__productos:
        p.imprimirProducto()

  elif  tipo =="compras":
    print("======== Listado de compras ========")
    for c in self.__compras:
        c.imprimirCompra()

def crear(self, tipo): # -----------------------------------------------------> Crear
  print("\n")
  if tipo =="clientes":
    print("=== Creacion de cliente ===")
    try:
      # Tomar valores del usuario
      codigo = input("Codigo: ")
      CI = input("CI: ")
      nombre = input("Nombre: ")
      correoE = input("Correo electronico: ")
      puntos = input("Puntos acumulados: ")

      # Convertir fechas de lenguaje natural a formato fijo
      fechaNac = Fecha(0,0,0)
      fechaNac.parseFecha(input("Fecha de nacimiento (DD/MM/YYYY): "))
      fechaIng = Fecha(0,0,0)
      fechaIng.parseFecha(input("Fecha de ingreso (DD/MM/YYYY): "))

      # Crear objeto cliente
      cliente = Cliente(codigo,
                        CI,
                        nombre,
                        fechaNac,
                        fechaIng,
                        correoE,
                        float(puntos))

      cliente.imprimirCliente() # Imprimir instancia cliente
      self.__clientes.append(cliente) # Agregar instancia a la lista
```

```
    print("Cliente creado exitosamente.")

  # Imprimir en caso de error
  except ValueError as e:
    print(f"Error creando Cliente: {e}.")


elif  tipo =="vendedores":
  # Obtencion y validacion de datos para creacion de vendedores
  print("=== Creacion de vendedor ===")
  try:
      # Toma de datos generales
      codigo = input("Codigo: ")
      CI = input("CI: ")
      nombre = input("Nombre: ")
      correoE = input("Correo electronico: ")
      sueldo = input("Sueldo: ")

      # Guardar horarios separados por coma
      horarios_str = input("Horarios (separados por coma): ")
      horarios = [h.strip() for h in horarios_str.split(',')]

      # Conversion de fecha de lenguaje natural
      fechaNac = Fecha(0,0,0)
      fechaNac.parseFecha(input("Fecha de nacimiento (DD/MM/YYYY): "))
      fechaIng = Fecha(0,0,0)
      fechaIng.parseFecha(input("Fecha de ingreso (DD/MM/YYYY): "))

      # Crear instancia de Vendedor
      vendedor = Vendedor(codigo,
                          CI,
                          nombre,
                          fechaNac,
                          fechaIng,
                          correoE,
                          float(sueldo),
                          horarios)

      vendedor.imprimirVendedor() # Imprimir instancia
      self.__vendedores.append(vendedor) # Agregar instancia a la lista
      print("Vendedor creado exitosamente.")

  # En caso de errores
  except ValueError as e:
      print(f"Error creando Vendedor: {e}.")

elif tipo == "productos":
  # Obtencion y validacion de datos para creacion de productos
  print("=== Creacion de producto ===")

  try:
    # Datos generales
    codigo = input("Codigo: ")
    nombre = input("Nombre: ")
    marca = input("Marca: ")
    modelo = input("Modelo: ")
    cantidad = input("Cantidad: ")
    unidad = input("Unidad: ")
    precio = input("Precio: ")
    activo_str = input("Activo (s/n): ").lower().strip()
```

```
        # Conversion de fecha
        fechaFab = Fecha(0, 0, 0)
        fechaFab.parseFecha(input("Fecha de fabricacion (DD/MM/YYYY): "))
        fechaVen = Fecha(0, 0, 0)
        fechaVen.parseFecha(input("Fecha de vencimiento (DD/MM/YYYY): "))

        # Instancia de producto
        producto = Producto(
            codigo=codigo,
            nombre=nombre,
            marca=marca,
            modelo=modelo,
            cantidad=float(cantidad),
            unidad=unidad,
            precio=float(precio),
            activo=(activo_str == "s"),
            fechaVen=fechaVen,
            fechaFab=fechaFab
        )

        producto.imprimirProducto() # Imprimir instancia
        self.__productos.append(producto) # Agregar instancia a la lista
        print("Producto creado exitosamente.")

      # En caso de errores
      except ValueError as e:
        print(f"Error creando producto: {e}. Verifique los valores ingresados.")

    self.run()

  def borrar(self, tipo): # ----------------------------------------------------> Borrar
    print("\n")

    if tipo =="clientes":
      # Obtencion y validacion de datos para eliminacion de clientes
      print("=== Borrado de clientes ===")
      self.listar("clientes")
      codigo = input("\nCodigo del cliente a borrar: ")
      found_cliente = None
      for cliente in self.__clientes[:]:
        if isinstance(cliente, Cliente):
            if cliente.getCodigo() == codigo:
              found_cliente = cliente
              break
        else:
             print(f"Adventencia: Saltando item invalido: {cliente}")


      if found_cliente:
          # Eliminar cliente de la lista y borrar datos
          self.__clientes.remove(found_cliente)
          found_cliente.eliminarDatosCliente()
          print("Cliente eliminado.")
          self.listar("clientes")
      else:
          print("Cliente no encontrado.")


    elif  tipo =="vendedores":
      # Obtencion y validacion de datos para eliminacion de vendedores
      print("=== Borrado de vendedores ===")
```

```
      self.listar("vendedores")
      codigo = input("\nCodigo del vendedor a borrar: ")
      found_vendedor = None
      for vendedor in self.__vendedores[:]:
        if isinstance(vendedor, Vendedor):
            if vendedor.getCodigo() == codigo:
              found_vendedor = vendedor
              break
        else:
             print(f"Adventencia: Saltando item invalido: {vendedor}")


      if found_vendedor:
          # Eliminar vendedor de la lista y borrar datos
          self.__vendedores.remove(found_vendedor)
          found_vendedor.eliminarDatosVendedor()
          print("Vendedor eliminado.")
          self.listar("vendedores")
      else:
          print("Vendedor no encontrado.")


    elif  tipo =="productos":
      # Obtencion y validacion de datos para eliminacion de productos
      print("=== Borrado de productos ===")
      self.listar("productos")
      codigo = input("\nCodigo del producto a borrar: ")
      found_producto = None
      for producto in self.__productos[:]:
        if isinstance(producto, Producto):
            if producto.getCodigo() == codigo:
              found_producto = producto
              break
        else:
             print(f"Adventencia: Saltando item invalido: {producto}")


      if found_producto:
          # Eliminar vendedor de la lista y borrar datos
          self.__productos.remove(found_producto)
          found_producto.eliminarDatosProducto()
          print("Producto eliminado.")
          self.listar("productos")
      else:
          print("Producto no encontrado.")

    self.run()

  def __buscar(self, tipo, codigo): # ----------------------------------------> __buscar()
    # Metodo privado para la busqueda de elementos a traves de sus codigos
    if tipo == "clientes":
      for cliente in self.__clientes:
        if isinstance(cliente, Cliente):
            if cliente.getCodigo() == codigo:
              return cliente
        else:
             print(f"Adventencia: Saltando item invalido: {cliente}")

    elif tipo == "vendedores":
      for vendedor in self.__vendedores:
        if isinstance(vendedor, Vendedor):
```

```
            if vendedor.getCodigo() == codigo:
               return vendedor
        else:
             print(f"Adventencia: Saltando item invalido: {vendedor}")

    elif tipo == "productos":
      for producto in self.__productos:
        if isinstance(producto, Producto):
            if producto.getCodigo() == codigo:
              return producto
        else:
             print(f"Adventencia: Saltando item invalido: {producto}")

    elif tipo == "compras":
      for compra in self.__compras:
        if isinstance(compra, Compra):
            if compra.getCodigo() == codigo:
              return compra
        else:
             print(f"Adventencia: Saltando item invalido: {compra}")

    return None


def compra(self, tipo): # ----------------------------------------------------> Crear
  Compra
  print("\n") # Logica programatica de compra
  if tipo == "crear":
      print("=== Creacion de compras ===")
      codigo = input("Codigo: ")

      # Convertir de lenguaje natural a formato consistente
      fechaCompra = Fecha(0, 0, 0)
      try:
          fechaCompra.parseFecha(input("Fecha de compra (DD/MM/AAAA): "))
      except ValueError as e:
          print(f"Error creando Compra: Formato de Fecha invalido - {e}")
          self.menuComprar()
          return
      # nFactura y modoPago
      nFactura = input("Numero de factura: ")
      modoPago = input("Modo de pago (ej: efectivo, tarjeta): ")

      # Convertir de lenguaje natural a formato consistente
      fechaVencFactura = Fecha(0, 0, 0)
      try:
          fechaVencFactura.parseFecha(input("Fecha de vencimiento factura (DD/MM/AAAA): ")
  )
      except ValueError:
          fechaVencFactura = None

      # Especificar cliente
      cliente_codigo = input("Codigo cliente: ")
      cliente = self.__buscar("clientes", cliente_codigo)
      if not isinstance(cliente, Cliente):
          print("Cliente no encontrado o invalido.")
          self.menuComprar()
          return

      # Especificar vendedor
      vendedor_codigo = input("Codigo vendedor: ")
```

```
vendedor = self.__buscar("vendedores", vendedor_codigo)
if not isinstance(vendedor, Vendedor):
    print("Vendedor no encontrado o invalido.")
    self.menuComprar()
    return

# Especificar items (productos y cantidades)
items = []
listado = []
while True:
    producto_codigo = input("Producto (Codigo): ")
    producto = self.__buscar("productos", producto_codigo)
    if not isinstance(producto, Producto):
        print("No agregado: producto no encontrado o invalido.")
        continue

    try:
        # Validacion de cantidades
        cantidad = int(input("Cantidad: "))
        if cantidad <= 0:
            print("La cantidad tiene que ser positiva.")
            continue
        if producto.getCantidad() < cantidad:
            print("No hay suficiente stock.")
            continue
        item = ItemCompra(producto, cantidad)
        items.append(item)
        listado.append(producto)
    except ValueError:
        print("Cantidad invalida. Por favor ingrese un numero.")
        continue

    if input("Desea agregar otro item? (s/n): ").lower() == "n":
        break
# Ultima verificacion
if len(items) == 0:
    print("No se agregaron items a la compra.")
    self.menuComprar()
    return

# Calcular total de compra
total = sum(it.getSubtotal() for it in items)

compra = Compra(
    codigo=codigo,
    fechaCompra=fechaCompra,
    cliente=cliente,
    vendedor=vendedor,
    items=items,
    nFactura=nFactura,
    fechaVencFactura=fechaVencFactura,
    modoPago=modoPago,
    listado=listado,
    valido=True,
    total=total
)

compra.imprimirCompra() # Imprimir objeto
self.__compras.append(compra) # Agregar objeto a la lista
print("Compra creada exitosamente.")
```

```
    elif  tipo =="suspender":
      print("=== Suspencion de compras ===")
      # Leer codigo de compra
      codigo = input("Codigo de la compra a suspender: ")
      found_compra = None
      for compra in self.__compras[:]:
        # Verificar que compra sea una instancia de Compra
        if isinstance(compra, Compra):
            if compra.getCodigo() == codigo:
              found_compra = compra
              break
        else:
             print(f"Adventencia: Saltando item invalido: {compra}")


      if found_compra:
          self.__compras.remove(found_compra) # Quitar compra de la lista
          found_compra.suspenderCompra() # Suspender compra y reponer stock
          print("Compra suspendida.")
      else:
          print("Compra no encontrada.")

    self.run()

# Fecha

class Fecha:
  def __init__(self,
               dia: int,
               mes: int,
               anio: int):
    # Atributos
    self.__dia = dia
    self.__mes = mes
    self.__anio = anio

  # Getters
  def getDia(self) -> int:
    return self.__dia

  def getMes(self) -> int:
    return self.__mes

  def getAnio(self) -> int:
    return self.__anio

  # Convertidor de string de fecha en lenguaje natural
  # a formato fijo, usando parser y dateparser
  def parseFecha(self, text: str):
    fecha =  self.__parse_to_date(text).strftime("%d/%m/%Y")
    self.__dia = fecha.split("/")[0]
    self.__mes = fecha.split("/")[1]
    self.__anio = fecha.split("/")[2]

  # Funcion utilizada por parseFecha
  def __parse_to_date(self, s: str):
    try:
        return parser.parse(s).date()
    except Exception:
        dt = dateparser.parse(s)
```

```
        if dt is None:
            raise ValueError(f"Could not parse date string: {s}")
        return dt.date()

  # Retornar los atributos como un string de fecha
  def toString(self) -> str:
    return f"{self.__dia}/{self.__mes}/{self.__anio}"

  # Eliminar fecha
  def eliminarFecha(self) -> bool:
    self.__dia = 0
    self.__mes = 0
    self.__anio = 0
    return True

  # Retornar string con formato ISO
  def to_iso(self) -> str:
    d = int(self.getDia())
    m = int(self.getMes())
    y = int(self.getAnio())
    return f"{y:04d}-{m:02d}-{d:02d}"

  # Retornar diccionario de fecha en ISO
  def to_dict(self) -> dict:
      return {"iso": self.to_iso()}

  # Leer fechas del diccionario
  @classmethod
  def from_dict(cls, d: dict) -> "Fecha":
      if isinstance(d, dict) and "iso" in d:
          y, m, d = map(int, d["iso"].split("-"))
          return cls(d, m, y)
      if isinstance(d, str):
          y, m, d = map(int, d.split("-"))
          return cls(d, m, y)
      raise ValueError(f"Formato invalido de fecha: {d}")

# Persona

class Persona:
  def __init__(self,
               codigo: str,
               CI: str,
               nombre: str,
               fechaNac: Fecha,
               fechaIng: Fecha,
               correoE: str):

    # Atributos
    self.__codigo = codigo      # privado
    self.__CI = CI              # privado
    self.__nombre = nombre      # privado
    self.__fechaNac = fechaNac  # privado
    self.__fechaIng = fechaIng  # privado
    self.__correoE = correoE    # privado

  # Getters
  def getCodigo(self) -> str:
    return self.__codigo

  def getCI(self) -> str:
```

```
    return self.__CI

  def getNombre(self) -> str:
    return self.__nombre

  def getFechaNac(self) -> Fecha:
    return self.__fechaNac

  def getFechaIng(self) -> Fecha:
    return self.__fechaIng

  def getCorreoE(self) -> str:
    return self.__correoE

  # ----- Otros metodos -----
  def imprimirPersona(self) -> None:
      print("\n======== Persona ==========")
      print(f"Codigo: {self.__codigo}")
      print(f"CI: {self.__CI}")
      print(f"Nombre: {self.__nombre}")
      print(f"Fecha de Nacimiento: {self.__fechaNac.toString()}")
      print(f"Fecha de Ingreso: {self.__fechaIng.toString()}")
      print(f"Correo: {self.__correoE}")

  def eliminarPersona(self) -> None:
      self.__codigo = ""
      self.__CI = ""
      self.__nombre = ""
      self.__fechaNac = self.__fechaNac.eliminarFecha()
      self.__fechaIng = self.__fechaIng.eliminarFecha()
      self.__correoE = ""

  # Leer fechas del diccionario
  def _fechas_from(d: dict) -> tuple["Fecha", "Fecha"]:
      y1, m1, d1 = map(int, d["fechaNac"].split("-"))
      fechaNac = Fecha(d1, m1, y1)
      y2, m2, d2 = map(int, d["fechaIng"].split("-"))
      fechaIng = Fecha(d2, m2, y2)
      return fechaNac, fechaIng

  # Retornar diccionario de Persona
  def to_persona_dict(self) -> dict:
    return {
      "codigo": self.getCodigo(),
      "CI": self.getCI(),
      "nombre": self.getNombre(),
      "fechaNac": self.getFechaNac().to_iso(),
      "fechaIng": self.getFechaIng().to_iso(),
      "correoE": self.getCorreoE(),
    }



# Vendedor

class Vendedor(Persona):
  def __init__(self,
               codigo: str,
               CI: str,
               nombre: str,
```

```
                fechaNac: Fecha,
                fechaIng: Fecha,
                correoE: str,
                sueldo: float,
                horarios: list):

    super().__init__(codigo, CI, nombre, fechaNac, fechaIng, correoE)
    self.__sueldo = sueldo
    self.__horarios = horarios  # lista de horarios

  def getSueldo(self) -> float:
    return self.__sueldo

  def getHorarios(self) -> list:
    return self.__horarios

  def imprimirVendedor(self) -> None:
    print("\n======== Vendedor ==========")
    self.imprimirPersona()
    print(f"\nSueldo: {self.__sueldo}")
    print(f"Horarios: {self.__horarios}")

  def eliminarDatosVendedor(self) -> None:
    self.sueldo = 0.0
    self.horarios = []
    self.eliminarPersona()
    print("Datos del vendedor eliminados.")

  # Retornar diccionario de atributos
  def to_dict(self) -> dict:
    base = self.to_persona_dict()
    base["sueldo"] = self.getSueldo()
    base["horarios"] = list(self.getHorarios())
    return base

  # Retornar atributos de diccionario
  @classmethod
  def from_dict(cls, d: dict) -> "Vendedor":
      fN, fI = Persona._fechas_from(d)
      return cls(
          d["codigo"], d["CI"], d["nombre"],
          fN, fI, d["correoE"], float(d["sueldo"]), list(d.get("horarios", []))
      )

# Cliente

class Cliente(Persona):
  def __init__(self,
                codigo: str,
                CI: str,
                nombre: str,
                fechaNac: Fecha,
                fechaIng: Fecha,
                correoE: str,
                puntos: float):

      super().__init__(codigo, CI, nombre, fechaNac, fechaIng, correoE)
      self.__puntos = puntos
  # Getters
  def getPuntos(self) -> float:
    return self.__puntos
```

```
  # Metodos de utilidad
  def imprimirCliente(self) -> None:
    print("\n======== Cliente ==========")
    self.imprimirPersona()
    print(f"\nPuntos acumulados: {self.__puntos}")

  def eliminarDatosCliente(self) -> bool:
    self.__puntos = 0.0
    self.eliminarPersona()
    print("Datos del cliente eliminados.")

  # Retornar diccionario de atributos
  def to_dict(self) -> dict:
    base = self.to_persona_dict()
    base["puntos"] = self.getPuntos()
    return base

  # Retornar atributos de diccionario
  @classmethod
  def from_dict(cls, d: dict) -> "Cliente":
      fN, fI = Persona._fechas_from(d)
      return cls(
          d["codigo"], d["CI"], d["nombre"],
          fN, fI, d["correoE"], float(d.get("puntos", 0.0))
      )

# Producto

class Producto:
    def __init__(self,
                 codigo: str,
                 nombre: str,
                 marca: str,
                 modelo: str,
                 cantidad: float,
                 unidad: str,
                 precio: float,
                 activo: bool,
                 fechaVen: Fecha,
                 fechaFab: Fecha):
        self.__codigo = codigo
        self.__nombre = nombre
        self.__marca = marca
        self.__modelo = modelo
        self.__cantidad = cantidad
        self.__unidad = unidad
        self.__precio = precio
        self.__activo = activo
        self.__fechaVen = fechaVen
        self.__fechaFab = fechaFab

    def getCodigo(self) -> str:
        return self.__codigo

    def getNombre(self) -> str:
        return self.__nombre

    def getMarca(self) -> str:
        return self.__marca
```

```
def getModelo(self) -> str:
    return self.__modelo

def getCantidad(self) -> float:
    return self.__cantidad

def getUnidad(self) -> str:
    return self.__unidad

def getPrecio(self) -> float:
    return self.__precio

def isActivo(self) -> bool:
    return self.__activo

def getFechaVen(self) -> Fecha:
    return self.__fechaVen

def getFechaFab(self) -> Fecha:
    return self.__fechaFab

def setCantidad(self, cantidad: float) -> None:
    self.__cantidad = cantidad

def imprimirProducto(self) -> None:
    print("\n======== Producto ==========")
    print(f"Codigo: {self.__codigo}")
    print(f"Nombre: {self.__nombre}")
    print(f"Marca: {self.__marca}")
    print(f"Modelo: {self.__modelo}")
    print(f"Cantidad: {self.__cantidad} {self.__unidad}")
    print(f"Precio: {self.__precio}")
    print(f"Activo: {'Si' if self.__activo else 'No'}")
    print(f"Fabricacion: {self.__fechaFab.toString() if self.__fechaFab else '-'}")
    print(f"Vencimiento: {self.__fechaVen.toString() if self.__fechaVen else '-'}")

def eliminarDatosProducto(self) -> bool:
  self.__codigo = ""
  self.__nombre = ""
  self.__marca = ""
  self.__modelo = ""
  self.__cantidad = 0.0
  self.__unidad = ""
  self.__precio = 0.0
  self.__activo = False
  self.__fechaVen = None
  self.__fechaFab = None
  print("\nDatos del producto eliminados.")
  return True

def to_dict(self) -> dict:
  return {
    "codigo": self.getCodigo(),
    "nombre": self.getNombre(),
    "marca": self.getMarca(),
    "modelo": self.getModelo(),
    "cantidad": self.getCantidad(),
    "unidad": self.getUnidad(),
    "precio": self.getPrecio(),
    "activo": self.isActivo(),
    "fechaVen": self.getFechaVen().to_dict() if self.__fechaVen else None,
```

```
      "fechaFab": self.getFechaFab().to_dict() if self.__fechaFab else None
    }

  @classmethod
  def from_dict(cls, d: dict) -> "Producto":
    fechaVen = Fecha.from_dict(d["fechaVen"]) if d.get("fechaVen") else None
    fechaFab = Fecha.from_dict(d["fechaFab"]) if d.get("fechaFab") else None
    return cls(
      d["codigo"],
      d["nombre"],
      d["marca"],
      d["modelo"],
      float(d["cantidad"]),
      d["unidad"],
      float(d["precio"]),
      bool(d["activo"]),
      fechaVen,
      fechaFab
    )

# ItemCompra

class ItemCompra:
  def __init__(self,
               producto: Producto,
               cantidad: int):
    self.__producto = producto
    self.__cantidad = cantidad

  def getProducto(self) -> Producto:
    return self.__producto

  def getCantidad(self) -> int:
    return self.__cantidad

  def getPrecio(self) -> float:
    return self.__producto.getPrecio()

  def getSubtotal(self) -> float:
    return self.getPrecio() * self.__cantidad

  def imprimirItemCompra(self) -> None:
    print("\n======== ItemCompra ==========")
    print(f"- Producto: ")
    self.__producto.imprimirProducto()
    print(f"\nCantidad: {self.__cantidad}")
    print(f"- Precio unitario: {self.__producto.getPrecio()}")
    print(f"- Subtotal: {self.getPrecio() * self.__cantidad}")

  def eliminarItemCompra(self) -> bool:
    self.__producto = None
    self.__cantidad = 0
    print("\nDatos del item de compra eliminados.")
    return True

  def to_record(self) -> dict:
    return {
      "producto_codigo": self.getProducto().getCodigo(),
      "cantidad": self.getCantidad(),
    }
```

```
  @classmethod
  def from_record(cls, rec: dict, productos_by_codigo: dict[str, Producto]) -> "ItemCompra":
    prod = productos_by_codigo.get(rec["producto_codigo"])
    if prod is None:
      raise ValueError(f"Producto {rec['producto_codigo']} no existe (al cargar compra)")
    return cls(prod, int(rec["cantidad"]))

# Compra

class Compra:
  def __init__(self,
               codigo: str,
               fechaCompra: "Fecha",
               cliente: "Cliente",
               vendedor: "Vendedor",
               items: list["ItemCompra"],
               nFactura: str = "",
               fechaVencFactura: "Fecha" = None,
               modoPago: str = "",
               listado: list["Producto"] = None,
               valido: bool = False,
               total: float = 0.0):
    self.__codigo = codigo
    self.__fechaCompra = fechaCompra
    self.__cliente = cliente
    self.__vendedor = vendedor
    self.__items = list(items) if items else []
    self.__nFactura = nFactura
    self.__fechaVencFactura = fechaVencFactura
    self.__modoPago = modoPago
    self.__listado = listado if listado else []
    self.__valido = valido
    self.__total = total

    # actualizar stock
    for item in self.__items:
      producto = item.getProducto()
      producto.setCantidad(producto.getCantidad() - item.getCantidad())
      print(f"Stock del producto {producto.getNombre()} actualizado.")

  # ---- Getters -----
  def getCodigo(self) -> str:
    return self.__codigo

  def getFecha(self) -> "Fecha":
    return self.__fechaCompra

  def getCliente(self) -> "Cliente":
    return self.__cliente

  def getVendedor(self) -> "Vendedor":
    return self.__vendedor

  def getItems(self) -> list["ItemCompra"]:
    return list(self.__items)

  def getFactura(self) -> str:
    return self.__nFactura

  def getFechaVencFactura(self) -> "Fecha":
    return self.__fechaVencFactura
```

```
def getModoPago(self) -> str:
  return self.__modoPago

def getListado(self) -> list["Producto"]:
  return list(self.__listado)

def esValida(self) -> bool:
  return self.__valido

def getTotal(self) -> float:
    if not self.__valido:
        return sum(it.getSubtotal() for it in self.__items)
    return self.__total

def suspenderCompra(self) -> None:
  self.__valido = False
  # Actualizar stock de productos
  for item in self.__items:
    producto = item.getProducto()
    producto.setCantidad(producto.getCantidad() + item.getCantidad())
    print(f"Stock del producto {producto.getNombre()} actualizado.")

def imprimirCompra(self) -> None:
  estado = "VaLIDA" if self.__valido else "ANULADA"
  print("\n======== Compra ==========")
  print(f"Codigo de compra: {self.__codigo}")
  print(f"Fecha: {self.__fechaCompra.toString()}")
  print(f"Factura: {self.__nFactura or '-'}  |  Pago: {self.__modoPago or '-'}  |  Vence:
  {self.__fechaVencFactura.toString() if self.__fechaVencFactura else '-'}")
  print(f"Estado: {estado}")
  print("Cliente:")
  self.__cliente.imprimirCliente()
  print("\nVendedor:")
  self.__vendedor.imprimirVendedor()
  print("\nItems:")
  for it in self.__items:
    it.imprimirItemCompra()
  print(f"\nTotal: {self.getTotal():.2f}")

# Retornar diccionario de atributos
def to_record(self) -> dict:
  return {
      "codigo": self.getCodigo(),
      "fecha": self.getFecha().to_iso(),
      "cliente_codigo": self.getCliente().getCodigo(),
      "vendedor_codigo": self.getVendedor().getCodigo(),
      "items": [it.to_record() for it in self.getItems()],
      "nFactura": self.getFactura(),
      "modoPago": self.getModoPago(),
      "fechaVencFactura": self.getFechaVencFactura().to_iso() if self.getFechaVencFactura
  () else None,
      "listado": [p.getCodigo() for p in self.getListado()],
      "valido": self.esValida(),
      "total": float(self.getTotal()),
  }

# Retornar atributos de diccionario
@classmethod
def from_record(
    cls,
```

```
        rec: dict,
        clientes_by_codigo: dict[str, "Cliente"],
        vendedores_by_codigo: dict[str, "Vendedor"],
        productos_by_codigo: dict[str, "Producto"],
    ) -> "Compra":
        fecha = Fecha(*map(int, rec["fecha"].split("-")[::-1]))
        cliente = clientes_by_codigo.get(rec["cliente_codigo"])
        vendedor = vendedores_by_codigo.get(rec["vendedor_codigo"])

        if cliente is None:
            raise ValueError(f"Cliente {rec['cliente_codigo']} no existe (al cargar compra)")
        if vendedor is None:
            raise ValueError(f"Vendedor {rec['vendedor_codigo']} no existe (al cargar compra)"
    )

        items = [ItemCompra.from_record(r, productos_by_codigo) for r in rec.get("items", [])]
        listado = [productos_by_codigo[cod] for cod in rec.get("listado", []) if cod in
    productos_by_codigo]
        fechaVencFactura = Fecha(*map(int, rec["fechaVencFactura"].split("-")[::-1])) if rec.
    get("fechaVencFactura") else None

        return cls(
            codigo=rec["codigo"],
            fechaCompra=fecha,
            cliente=cliente,
            vendedor=vendedor,
            items=items,
            nFactura=rec.get("nFactura", ""),
            fechaVencFactura=fechaVencFactura,
            modoPago=rec.get("modoPago", ""),
            listado=listado,
            valido=bool(rec.get("valido", False)),
            total=float(rec.get("total", 0.0)),
        )

# Valores de prueba

# Fechas
f1 = Fecha(1, 1, 1990)
f2 = Fecha(15, 6, 1985)
f3 = Fecha(10, 10, 2000)
f4 = Fecha(5, 5, 1995)
f5 = Fecha(20, 3, 2024)
f6 = Fecha(20, 4, 2024)

# Clientes
c1 = Cliente(CI="11111111", codigo="01", correoE="maria@mail.com", fechaIng=f1, fechaNac=f2,
     nombre="Maria", puntos=10.0)
c2 = Cliente(CI="22221111", codigo="02", correoE="carlos@mail.com", fechaIng=f3, fechaNac=f4
    , nombre="Carlos", puntos=300.0)
listaClientes = [c1, c2]

# Vendedores
v1 = Vendedor(CI="1111235123", codigo="01", correoE="juan@mail.com", fechaIng=f1, fechaNac=
    f2, nombre="Juan", horarios=["9-18"], sueldo=123.2)
v2 = Vendedor(CI="123123123123", codigo="02", correoE="diego@mail.com", fechaIng=f3,
    fechaNac=f5, nombre="Diego", horarios=["10-19"], sueldo=1231233.2)
listaVendedores = [v1, v2]

# Productos
p1 = Producto(codigo="P00", nombre="Laptop", marca="Dell", modelo="Inspiron", cantidad=100,
```

```
    unidad="unid", precio=1000.1, activo=True, fechaVen=f6, fechaFab=f1)
p2 = Producto(codigo="P01", nombre="Tablet", marca="Samsung", modelo="Tab A", cantidad=200,
    unidad="unid", precio=500.5, activo=True, fechaVen=f6, fechaFab=f2)
p3 = Producto(codigo="P02", nombre="Celular", marca="Apple", modelo="iPhone", cantidad=300,
    unidad="unid", precio=200.2, activo=True, fechaVen=f6, fechaFab=f3)
listaProductos = [p1, p2, p3]

# ItemCompra
i1 = ItemCompra(cantidad=2, producto=p1)   # 2 Laptops
i2 = ItemCompra(cantidad=10, producto=p2)  # 10 Tablets
i3 = ItemCompra(cantidad=20, producto=p3)  # 20 Celulares

# Compras
compra1 = Compra(
    codigo="C01",
    nFactura="F001",
    fechaCompra=f1,
    fechaVencFactura=f6,
    modoPago="Tarjeta",
    listado=[p1, p2, p3],
    valido=True,
    total=i1.getSubtotal() + i2.getSubtotal() + i3.getSubtotal(),
    cliente=c1,
    vendedor=v1,
    items=[i1, i2, i3]
)

compra2 = Compra(
    codigo="C02",
    nFactura="F002",
    fechaCompra=f5,
    fechaVencFactura=f6,
    modoPago="Efectivo",
    listado=[p2, p2, p2],
    valido=True,
    total=(i2.getSubtotal() * 3),  # 3 veces el mismo item
    cliente=c2,
    vendedor=v2,
    items=[i2, i2, i2]
)

listaCompras = [compra1, compra2]

# Guardar
save_all(
    "./db",
    listaClientes,
    listaVendedores,
    listaProductos,
    listaCompras
)

# Resultado

menu = Menu(direccion="./db")
menu.run()
```

C. EVIDENCIAS DE FUNCIONAMIENTO

```
[22]  menu = Menu(direccion="./db")
      menu.run()

...   Stock del producto Laptop actualizado.
      Stock del producto Tablet actualizado.
      Stock del producto Celular actualizado.
      Stock del producto Tablet actualizado.
      Stock del producto Tablet actualizado.
      Stock del producto Tablet actualizado.


      ////////// Menu principal //////////
      1 - Consultar informacion
      2 - Crear informacion
      3 - Borrar informacion
      4 - Menu compras
      0 - Salir


      Ingrese una opcion:
```

Figura 4: Menú principal

```
////////// Menu crear //////////
1 - Crear cliente
2 - Crear vendedor
3 - Crear producto
0 - Volver


Ingrese una opcion:1


=== Creacion de cliente ===
Codigo: 05
CI: 58213893
Nombre: Juan
Correo electronico: juan@mail.com
Puntos acumulados: 1023
Fecha de nacimiento (DD/MM/YYYY): 10 10 2010
Fecha de ingreso (DD/MM/YYYY): 10 05 2025

======== Cliente ==========

======== Persona ==========
Codigo: 05
CI: 58213893
Nombre: Juan
Fecha de Nacimiento: 10/10/2010
Fecha de Ingreso: 05/10/2025
Correo: juan@mail.com

Puntos acumulados: 1023.0
Cliente creado exitosamente.
```

Figura 5: Creación de cliente

```
////////// Menu crear //////////
1 - Crear cliente
2 - Crear vendedor
3 - Crear producto
0 - Volver


Ingrese una opcion:2


=== Creacion de vendedor ===
Codigo: 05
CI: 12301248
Nombre: Pepe
Correo electronico: pepe@mail.com
Sueldo: 1023
Horarios (separados por coma): 9,17
Fecha de nacimiento (DD/MM/YYYY): 12/09/2005
Fecha de ingreso (DD/MM/YYYY): 12/12/2012

======== Vendedor ==========

======== Persona ==========
Codigo: 05
CI: 12301248
Nombre: Pepe
Fecha de Nacimiento: 09/12/2005
Fecha de Ingreso: 12/12/2012
Correo: pepe@mail.com

Sueldo: 1023.0
Horarios: ['9', '17']
Vendedor creado exitosamente.
```

Figura 6: Creación de vendedor

```
////////// Menu crear //////////
1 - Crear cliente
2 - Crear vendedor
3 - Crear producto
0 - Volver


Ingrese una opcion:3


=== Creacion de producto ===
Codigo: P04
Nombre: Heladera
Marca: Epson
Modelo: Turno
Cantidad: 100
Unidad: 1
Precio: 2023
Activo (s/n): s
Fecha de fabricacion (DD/MM/YYYY): 10/10/10
Fecha de vencimiento (DD/MM/YYYY): 10/20/10

======== Producto ==========
Codigo: P04
Nombre: Heladera
Marca: Epson
Modelo: Turno
Cantidad: 100.0 1
Precio: 2023.0
Activo: Si
Fabricacion: 10/10/2010
Vencimiento: 20/10/2010
Producto creado exitosamente.
```

Figura 7: Creación de producto

```
////////// Consultar //////////
1 - Listar clientes
2 - Listar vendedores
3 - Listar productos
4 - Listar compras
0 - Volver


Ingrese una opcion:1


======== Listado de clientes ========

======== Cliente ==========

======== Persona ==========
Codigo: 01
CI: 11111111
Nombre: Maria
Fecha de Nacimiento: 15/6/1985
Fecha de Ingreso: 1/1/1990
Correo: maria@mail.com

Puntos acumulados: 10.0

======== Cliente ==========

======== Persona ==========
Codigo: 02
CI: 22221111
Nombre: Carlos
Fecha de Nacimiento: 5/5/1995
Fecha de Ingreso: 10/10/2000
Correo: carlos@mail.com

Puntos acumulados: 300.0
```

Figura 8: Listado de clientes

```
////////// Consultar //////////
1 - Listar clientes
2 - Listar vendedores
3 - Listar productos
4 - Listar compras
0 - Volver


Ingrese una opcion:2


======== Listado de vendedores ========

======== Vendedor ==========

======== Persona ==========
Codigo: 01
CI: 1111235123
Nombre: Juan
Fecha de Nacimiento: 15/6/1985
Fecha de Ingreso: 1/1/1990
Correo: juan@mail.com

Sueldo: 123.2
Horarios: ['9-18']

======== Vendedor ==========

======== Persona ==========
Codigo: 02
CI: 123123123123
Nombre: Diego
Fecha de Nacimiento: 20/3/2024
Fecha de Ingreso: 10/10/2000
Correo: diego@mail.com

Sueldo: 1231233.2
Horarios: ['10-19']
```

Figura 9: Listado de vendedores

```
////////// Consultar //////////
1 - Listar clientes
2 - Listar vendedores
3 - Listar productos
4 - Listar compras
0 - Volver


Ingrese una opcion:3


======== Listado de productos ========

======== Producto ==========
Codigo: P00
Nombre: Laptop
Marca: Dell
Modelo: Inspiron
Cantidad: 96.0 unid
Precio: 1000.1
Activo: Si
Fabricacion: 1/1/1990
Vencimiento: 20/4/2024

======== Producto ==========
Codigo: P01
Nombre: Tablet
Marca: Samsung
Modelo: Tab A
Cantidad: 120.0 unid
Precio: 500.5
Activo: Si
Fabricacion: 15/6/1985
Vencimiento: 20/4/2024

======== Producto ==========
Codigo: P02
Nombre: Celular
Marca: Apple
Modelo: iPhone
Cantidad: 260.0 unid
Precio: 200.2
Activo: Si
Fabricacion: 10/10/2000
Vencimiento: 20/4/2024
```

Figura 10: Listado de productos

```
////////// Menu comprar //////////
1 - Crear compra
2 - Suspender compra
0 - Volver


Ingrese una opcion:1


=== Creacion de compras ===
Codigo: 123
Fecha de compra (DD/MM/AAAA): 123
Numero de factura: 230
Modo de pago (ej: efectivo, tarjeta): Efectivo
Fecha de vencimiento factura (DD/MM/AAAA): 12 12 12
Codigo cliente: 02
Codigo vendedor: 02
Producto (Codigo): P02
Cantidad: 6
¿Desea agregar otro item? (s/n): s
Producto (Codigo): P05
No agregado: producto no encontrado o invalido.
Producto (Codigo): P04
Cantidad: 1
¿Desea agregar otro item? (s/n): n
Stock del producto Celular actualizado.
Stock del producto Heladera actualizado.

======== Compra ==========
Codigo de compra: 123
Fecha: 03/10/123
Factura: 230  |  Pago: Efectivo  |  Vence: 12/12/2012
Estado: VaLIDA
Cliente:

======== Cliente ==========

======== Persona ==========
Codigo: 02
CI: 22221111
Nombre: Carlos
Fecha de Nacimiento: 5/5/1995
Fecha de Ingreso: 10/10/2000
Correo: carlos@mail.com

Puntos acumulados: 300.0
```

Figura 11: Creación de compra (paso 1)

```
======== Persona ==========
Codigo: 02
CI: 123123123123
Nombre: Diego
Fecha de Nacimiento: 20/3/2024
Fecha de Ingreso: 10/10/2000
Correo: diego@mail.com

Sueldo: 1231233.2
Horarios: ['10-19']

Items:

======== ItemCompra ==========
- Producto:

======== Producto ==========
Codigo: P02
Nombre: Celular
Marca: Apple
Modelo: iPhone
Cantidad: 254.0 unid
Precio: 200.2
Activo: Si
Fabricacion: 10/10/2000
Vencimiento: 20/4/2024

Cantidad: 6
- Precio unitario: 200.2
- Subtotal: 1201.1999999999998

======== ItemCompra ==========
- Producto:

======== Producto ==========
Codigo: P04
Nombre: Heladera
Marca: Epson
Modelo: Turno
Cantidad: 99.0 1
Precio: 2023.0
Activo: Si
Fabricacion: 10/10/2010
Vencimiento: 20/10/2010

Cantidad: 1
- Precio unitario: 2023.0
- Subtotal: 2023.0

Total: 3224.20
Compra creada exitosamente.
```

Figura 12: Creación de compra (paso 2)

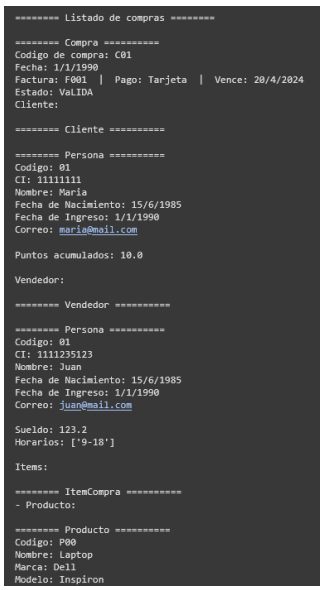

Figura 13: Listado de compras

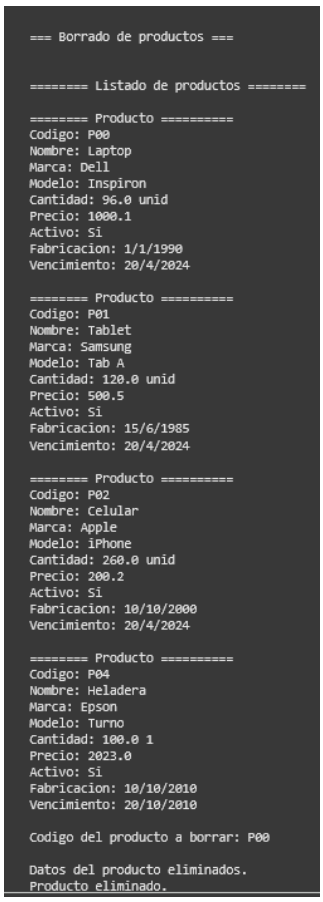

Figura 15: Borrado de producto

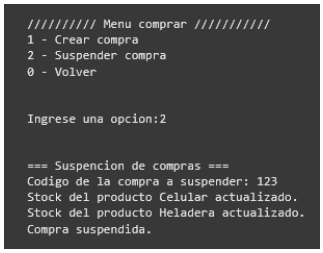

Figura 14: Suspensión de compra

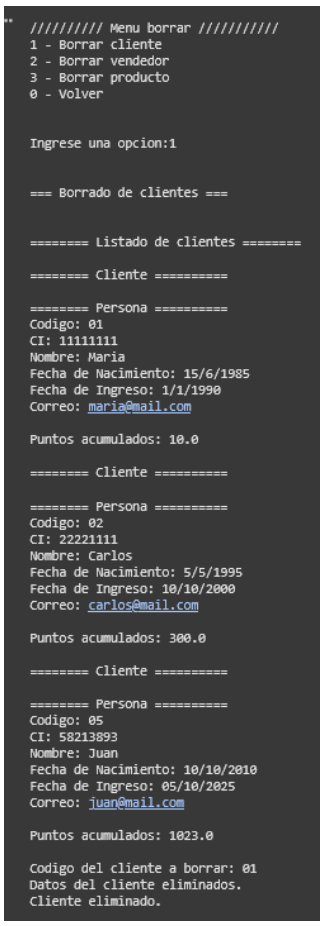

Figura 16: Borrado de cliente

Figura 17: Borrado de vendedor

D. RESULTADOS EN ARCHIVOS .TXT


```
{"CI": "22221111", "codigo": "02", "correoE": "carlos@mail.com", "fechaIng": "2000-10-10", "
    fechaNac": "1995-05-05", "nombre": "Carlos", "puntos": 300.0}
{"CI": "58213893", "codigo": "05", "correoE": "juan@mail.com", "fechaIng": "2025-10-05", "
    fechaNac": "2010-10-10", "nombre": "Juan", "puntos": 1023.0}

{"cliente_codigo": "", "codigo": "C01", "fecha": "1990-01-01", "fechaVencFactura":
    "2024-04-20", "items": [{"cantidad": 2, "producto_codigo": ""}, {"cantidad": 10, "
    producto_codigo": "P01"}, {"cantidad": 20, "producto_codigo": "P02"}], "listado": ["", "
    P01", "P02"], "modoPago": "Tarjeta", "nFactura": "F001", "total": 11009.2, "valido":
    true, "vendedor_codigo": ""}
{"cliente_codigo": "02", "codigo": "C02", "fecha": "2024-03-20", "fechaVencFactura":
    "2024-04-20", "items": [{"cantidad": 10, "producto_codigo": "P01"}, {"cantidad": 10, "
    producto_codigo": "P01"}, {"cantidad": 10, "producto_codigo": "P01"}], "listado": ["P01
    ", "P01", "P01"], "modoPago": "Efectivo", "nFactura": "F002", "total": 15015.0, "valido
    ": true, "vendedor_codigo": "02"}

{"CI": "123123123123", "codigo": "02", "correoE": "diego@mail.com", "fechaIng":
    "2000-10-10", "fechaNac": "2024-03-20", "horarios": ["10-19"], "nombre": "Diego", "
    sueldo": 1231233.2}
{"CI": "12301248", "codigo": "05", "correoE": "pepe@mail.com", "fechaIng": "2012-12-12", "
    fechaNac": "2005-12-09", "horarios": ["9", "17"], "nombre": "Pepe", "sueldo": 1023.0}

{"activo": true, "cantidad": 120.0, "codigo": "P01", "fechaFab": {"iso": "1985-06-15"}, "
    fechaVen": {"iso": "2024-04-20"}, "marca": "Samsung", "modelo": "Tab A", "nombre": "
    Tablet", "precio": 500.5, "unidad": "unid"}
{"activo": true, "cantidad": 260.0, "codigo": "P02", "fechaFab": {"iso": "2000-10-10"}, "
    fechaVen": {"iso": "2024-04-20"}, "marca": "Apple", "modelo": "iPhone", "nombre": "
    Celular", "precio": 200.2, "unidad": "unid"}
{"activo": true, "cantidad": 100.0, "codigo": "P04", "fechaFab": {"iso": "2010-10-10"}, "
    fechaVen": {"iso": "2010-10-20"}, "marca": "Epson", "modelo": "Turno", "nombre": "
    Heladera", "precio": 2023.0, "unidad": "1"}
```