TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE DURANGO

ING. SISTEMAS COMPUTACIONALES

PROGRAMACIÓN ORIENTADA A OBJETOS

PROYECTO UNIDAD 3- FERRETERIA

DOCENTE: RODRIGUEZ RIVAS JOSE GABRIEL

ALUMNOS:

MELENDEZ ARRIETA ITZEL ANDREA

CARRILLO FERRRER CARLA IBERIA

PASTRANA CANALES CESAR GUILLERMO

AVALOS ALVAREZ JOEL

DURANGO, DGO. 31 DE OCTUBRE DEL 2025

INDICE

Introduccion 3
Descripción del funcionamiento del programa 4
Clase Persona 4
Clase Cliente(Persona) 5
Clase Empleado(Persona) 6
Clase Producto 7
Clase InventarioProducto 8
Clase Venta 9
Control de sesión 10
Funciones adicionales 11
Funcionamiento del menú paso a paso 12
Inicio de sesión obligatorio: 12
Menú de opciones: 12
Selección de opción: 13
Operaciones principales del menú: 13
- Opción 1: 13
- Opción 2 14
- Opción 3: 14
- Opción 4 14
- Opción 5 14
- Opción 6 15
- Opción 7 15
- Opción 8: 15
- Opción 9 16
- Opción 10: 16

Control de flujo: 16
Cierre de sesión y salida: 16
Conclusión 17

Introduccion

En este documento explicaremos el desarrollo e implementación de un sistema de gestión para una ferretería, utilizando los principios de la Programación Orientada a Objetos (POO). Este proyecto, tiene como fin diseñar una estructura lógica para la administración de las entidades centrales del negocio: el personal, los clientes, el inventario y las transacciones de venta.

El objetivo primordial es aplicar el concepto de Herencia en POO, lo que representa un avance significativo en la estructuración y reutilización del código. Específicamente, se ha identificado que las entidades de Cliente y Empleado comparten datos básicos como el nombre, el teléfono y el correo electrónico. Por lo tanto, se ha creado una clase base, 'Clase Persona', de la cual heredan ambas clases, logrando organizar mejor el sistema y evitar la repetición de código.

Se incluye un mecanismo de inicio de sesión obligatorio para empleados, un menú de opciones para las operaciones principales (registro de productos, clientes y ventas), y diversas funciones de búsqueda y control que conectan todas las clases para asegurar el correcto funcionamiento integrado del programa.

Descripción del funcionamiento del programa

Clase Persona

Descripción:
Es la **clase base** que representa a cualquier persona dentro del sistema (cliente o empleado).
Se usa como modelo general del que heredan otras clases.

Atributos:

- nombre: Nombre completo de la persona.
- correo: Correo electrónico.
- direccion: Domicilio o dirección de contacto.
- telefono: Número de teléfono.

Métodos:

- __init__(): Constructor que inicializa los datos de la persona.
- mostrar_info(): Muestra en pantalla la información general (nombre, correo, dirección y teléfono).

Función principal:
Evita repetir código en las clases Cliente y Empleado, ya que ambos comparten los mismos datos básicos.

```python
class Persona:  2 usages
    def __init__(self, nombre, correo, direccion, telefono):
        self.nombre = nombre
        self.correo = correo
        self.direccion = direccion
        self.telefono = telefono

    def mostrar_info(self):  23 usages (21 dynamic)
        print(f"Nombre: {self.nombre} | Correo: {self.correo} |"
              f" Dirección: {self.direccion} | Teléfono: {self.telefono}")
```

Clase Cliente(Persona)

Descripción:
Representa a un **cliente del negocio**, hereda de la clase Persona y agrega el atributo **RFC**, que es importante para ventas y facturación.

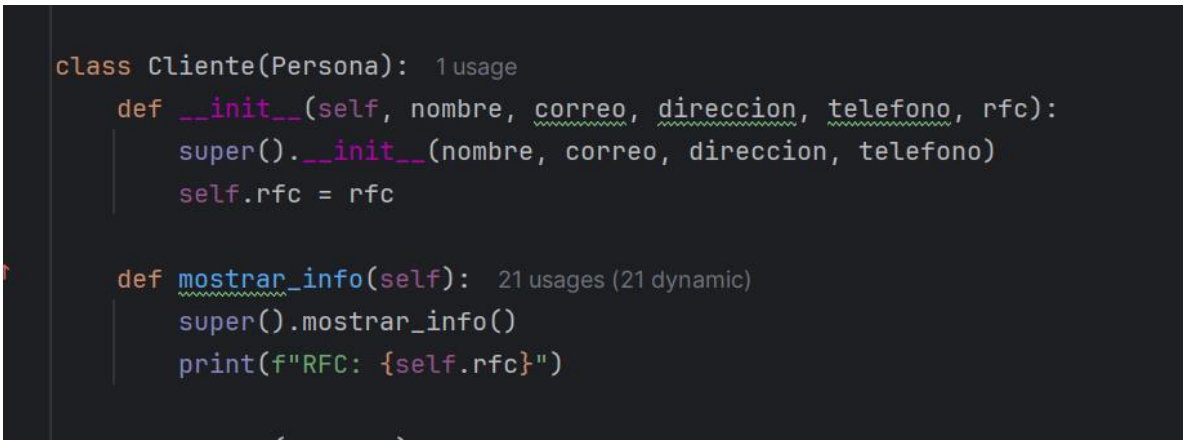
Atributos adicionales:

- rfc: Registro Federal de Contribuyentes, identificador fiscal del cliente.

Métodos:

- mostrar_info(): Muestra la información general de la persona y también el RFC.

Función principal:
Almacenar los datos de los clientes registrados y poder consultarlos o vincularlos a las ventas.



```python
class Cliente(Persona):  1 usage
    def __init__(self, nombre, correo, direccion, telefono, rfc):
        super().__init__(nombre, correo, direccion, telefono)
        self.rfc = rfc

    def mostrar_info(self):  21 usages (21 dynamic)
        super().mostrar_info()
        print(f"RFC: {self.rfc}")
```

Clase Empleado(Persona)

Descripción:
Representa a los **empleados del sistema**, también hereda de Persona y añade datos laborales y de acceso.

Atributos adicionales:

- id_empleado: Código único que identifica al empleado.
- departamento: Área en la que trabaja (por ejemplo, Administración, Ventas, etc.).
- usuario: Nombre de usuario para iniciar sesión.
- contrasena: Contraseña para acceder al sistema.

Métodos:

- mostrar_info(): Muestra los datos generales y el identificador del empleado con su departamento.

Función principal:
Permitir el **acceso al sistema** y mantener un registro básico del personal que opera el programa

```python
class Empleado(Persona):  1 usage
    def __init__(self, nombre, correo, direccion, telefono, id_empleado, departamento, usuario, contrasena):
        super().__init__(nombre, correo, direccion, telefono)
        self.id_empleado = id_empleado
        self.departamento = departamento
        self.usuario = usuario
        self.contrasena = contrasena

    def mostrar_info(self):  21 usages (21 dynamic)
        super().mostrar_info()
        print(f"ID Empleado: {self.id_empleado} | Departamento: {self.departamento}")
```

Clase Producto

Descripción:
Representa un **producto o artículo** disponible en el inventario.

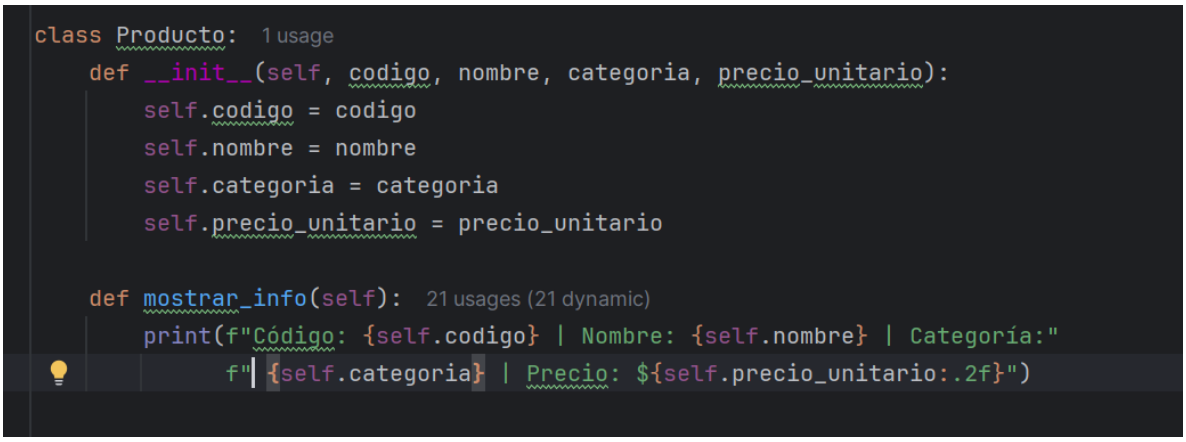
Atributos:

- codigo: Código único del producto.
- nombre: Nombre o descripción del producto.
- categoria: Tipo o clasificación (por ejemplo, herramientas, pintura, etc.).
- precio_unitario: Precio de venta por unidad.

Métodos:

- mostrar_info(): Muestra los datos del producto y su precio.

Función principal:
Identificar cada producto dentro del sistema y servir como base para el control del inventario y las ventas.



```python
class Producto:  1 usage
    def __init__(self, codigo, nombre, categoria, precio_unitario):
        self.codigo = codigo
        self.nombre = nombre
        self.categoria = categoria
        self.precio_unitario = precio_unitario

    def mostrar_info(self):  21 usages (21 dynamic)
        print(f"Código: {self.codigo} | Nombre: {self.nombre} | Categoría:"
              f" {self.categoria} | Precio: ${self.precio_unitario:.2f}")
```

Clase InventarioProducto

Descripción:
Administra el **inventario de cada producto**, controlando las existencias, las unidades vendidas y las disponibles.

Atributos:

- producto: Referencia a un objeto de tipo Producto.
- cantidad: Número total de unidades registradas en inventario.
- vendidos: Cantidad de productos que ya se han vendido.

Métodos:

- disponibles(): Calcula cuántas unidades quedan disponibles (cantidad - vendidos).

- vender(cantidad): Registra una venta, disminuyendo el inventario si hay suficiente stock.
- agregar_stock(cantidad): Aumenta la cantidad disponible del producto.
- mostrar_estado(): Muestra toda la información del producto junto con su estado en inventario (existencia, vendidos, disponibles).

Función principal:
Llevar el **control automático de las existencias** y evitar ventas sin stock.

```
class InventarioProducto:  1 usage
    def __init__(self, producto, cantidad):
        self.producto = producto
        self.cantidad = cantidad
        self.vendidos = 0

    def disponibles(self):  2 usages
        return self.cantidad - self.vendidos

    def vender(self, cantidad):
        if cantidad <= self.disponibles():
            self.vendidos += cantidad
            print(f"Venta registrada: {cantidad} unidades de {self.producto.nombre}")
            return True
        else:
            print("No hay suficiente inventario disponible.")
            return False

    def agregar_stock(self, cantidad):
        self.cantidad += cantidad
        print(f"Se agregaron {cantidad} unidades al inventario de {self.producto.nombre}")

    def mostrar_estado(self):  5 usages (5 dynamic)
        self.producto.mostrar_info()
        print(f"Existencia total: {self.cantidad} | Vendidos: {self.vendidos} | Disponibles: {self.disponibles()}")
```

Clase Venta

Descripción:
Representa una **transacción de venta** dentro del sistema.

Atributos:

- folio: Número único que identifica la venta.
- id_cliente: RFC del cliente que realizó la compra.
- codigo_producto: Código del producto vendido.
- fecha: Fecha y hora en que se realizó la venta.
- cantidad: Cantidad de unidades vendidas.
- total: Monto total de la venta (precio unitario × cantidad).

Métodos:

- mostrar_info(): Muestra todos los detalles de la venta.

Función principal:
Registrar cada transacción con su información completa para tener un **historial de ventas ordenado y verificable**.

```python
class Venta:  1 usage
    def __init__(self, folio, id_cliente, codigo_producto, fecha, cantidad, total):
        self.folio = folio
        self.id_cliente = id_cliente
        self.codigo_producto = codigo_producto
        self.fecha = fecha
        self.cantidad = cantidad
        self.total = total

    def mostrar_info(self):  21 usages (21 dynamic)
        print(f"Folio: {self.folio} | Cliente: {self.id_cliente} | Producto: {self.codigo_producto} "
              f"| Fecha: {self.fecha} | Cantidad: {self.cantidad} | Total: ${self.total:.2f}")
```

Control de sesión

- Los empleados deben **iniciar sesión** para usar el sistema.
- Se puede **cerrar sesión** o **salir completamente del programa**.

Este control básico de usuarios agrega un **nivel de seguridad y orden** en el manejo del sistema, en el cual solo con el usuario admin y la contraseña 1234, se podrá ingresar al programa.

```python
# Empleado administrador por defecto
empleados.append(Empleado( nombre: "Admin",  correo: "admin@ferre.com",  direccion: "Sucursal Central",  telefono: "555-0000"
                          ,  id_empleado: "E001",  departamento: "Administración",  usuario: "admin",  contrasena: "1234"))
```

Funciones adicionales

Además de las clases, el programa cuenta con diversas funciones que ayudan al funcionamiento del sistema:

- buscar_producto(codigo): Localiza un producto en el inventario.
- buscar_cliente(rfc): Busca un cliente por su RFC.
- login(): Permite a los empleados iniciar sesión con su usuario y contraseña.
- logout(): Cierra la sesión actual.
- menu(): Controla la interacción del usuario con el sistema a través de un menú de opciones.

Estas funciones conectan las clases entre sí, permitiendo que el sistema trabaje de forma integrada.

```python
def buscar_producto(codigo):  2 usages
    for item in inventario:
        if item.producto.codigo == codigo:
            return item
    return None

def buscar_cliente(id_cliente):  1 usage
    for cliente in clientes:
        if cliente.rfc == id_cliente:
            return cliente
    return None
```

```python
def login():  1 usage
    global sesion_activa
    print("\n INICIO DE SESIÓN")
    usuario = input("Usuario: ")
    contrasena = input("Contraseña: ")
    for emp in empleados:
        if emp.usuario == usuario and emp.contrasena == contrasena:
            sesion_activa = emp
            print(f"Bienvenido, {emp.nombre} ({emp.departamento})")
            return True
    print("Credenciales incorrectas.")
    return False

def logout():  1 usage
    global sesion_activa
    sesion_activa = None
    print("Sesión cerrada.")
```

Funcionamiento del menú paso a paso

Inicio de sesión obligatorio:

Antes de mostrar el menú, el sistema verifica si hay un usuario con sesión activa.
Si no la hay, se ejecuta la función login(), que solicita el nombre de usuario y la contraseña.
Solo un empleado con credenciales válidas puede acceder al menú principal.

Menú de opciones:

Una vez iniciada la sesión, el programa muestra un menú numerado con las siguientes opciones:

```python
def menu():  1 usage
    global folio_actual
    while True:
        if not sesion_activa:
            if not login():
                continue

        print("\n MENÚ PRINCIPAL")
        print("1. Dar de alta un producto")
        print("2. Mostrar inventario de productos")
        print("3. Consultar producto por código")
        print("4. Registrar cliente")
        print("5. Consultar cliente por RFC")
        print("6. Mostrar todos los clientes")
        print("7. Registrar una venta")
        print("8. Mostrar historial de ventas")
        print("9. Cerrar sesión")
        print("10. Salir")

        opcion = input("Seleccione una opción: ")
```

Selección de opción:

El usuario introduce el número correspondiente a la acción que desea realizar. Dependiendo del valor ingresado, se ejecuta un bloque if que llama a la función o clase correspondiente.

Operaciones principales del menú:

- Opción 1: Registrar un nuevo producto y añadirlo al inventario.

```python
    if opcion == "1":
        codigo = input("Código del producto: ")
        nombre = input("Nombre: ")
        categoria = input("Categoría: ")
        precio = float(input("Precio unitario: "))
        cantidad = int(input("Cantidad inicial: "))
        nuevo_producto = Producto(codigo, nombre, categoria, precio)
        productos.append(nuevo_producto)
        inventario.append(InventarioProducto(nuevo_producto, cantidad))
        print("Producto registrado.")
```

- Opción 2**:** Mostrar el estado de todos los productos, con su stock total, vendidos y disponibles.

```
elif opcion == "2":
    for item in inventario:
        item.mostrar_estado()
```

- Opción 3: Buscar un producto por su código y mostrar su información.

```
elif opcion == "3":
    codigo = input("Código del producto: ")
    item = buscar_producto(codigo)
    if item:
        item.mostrar_estado()
    else:
        print("Producto no encontrado.")
```

- Opción 4**:** Registrar un nuevo cliente con todos sus datos personales y su RFC.

```
elif opcion == "4":
    nombre = input("Nombre: ")
    correo = input("Correo: ")
    direccion = input("Dirección: ")
    telefono = input("Teléfono: ")
    rfc = input("RFC: ")
    clientes.append(Cliente(nombre, correo, direccion, telefono, rfc))
    print("Cliente registrado.")
```

- Opción 5**:** Consultar los datos de un cliente específico mediante su RFC.

```
elif opcion == "5":
    rfc = input("RFC del cliente: ")
    cliente = buscar_cliente(rfc)
    if cliente:
        cliente.mostrar_info()
    else:
        print("Cliente no encontrado.")
```

- Opción 6**:** Mostrar la lista completa de clientes registrados.

```
elif opcion == "6":
    for cliente in clientes:
        cliente.mostrar_info()
```

- Opción 7**:** Registrar una venta, verificando el stock del producto y generando un folio de venta.

```
elif opcion == "7":
    rfc = input("RFC del cliente (999999 para público general): ")
    codigo = input("Código del producto: ")
    cantidad = int(input("Cantidad a vender: "))
    item = buscar_producto(codigo)
    if item and item.vender(cantidad):
        total = item.producto.precio_unitario * cantidad
        fecha = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        venta = Venta(folio_actual, rfc, codigo, fecha, cantidad, total)
        ventas.append(venta)
        folio_actual += 1
        print("Venta registrada.")
    else:
        print("No se pudo completar la venta.")
```

- Opción 8: Mostrar el historial completo de ventas registradas.

```
elif opcion == "8":
    for venta in ventas:
        venta.mostrar_info()
```

- Opción 9**:** Cerrar sesión del empleado actual.

```
elif opcion == "9":
    logout()
```

- Opción 10: Salir completamente del sistema.

```
        elif opcion == "10":
            print("Gracias por usar el sistema. ¡Hasta pronto!")
            break

        else:
            print("Opción inválida. Intente de nuevo.")

menu()
```

Control de flujo:

Si el usuario introduce una opción no válida, el sistema muestra un mensaje de error y vuelve a mostrar el menú sin cerrar el programa.

Cierre de sesión y salida:

- Con la opción 9, se ejecuta la función logout(), que termina la sesión actual.
- Con la opción 10, se rompe el ciclo while, finalizando el programa con un mensaje de despedida.

Conclusión

En este proyecto vamos a ampliar el sistema de gestión para una ferretería haciendo uso de Herencia en POO, lo cual va representar un avance significativo en la estructuración y reutilización del código. Al notar que los clientes y empleados comparten datos similares como nombre, teléfono y correo, se pueden aprovechar esa coincidencia para organizar mejor el sistema y evitar repetir código para facilitar futuras mejoras o cambios. En resumen, al mejorar el sistema de ferretería con herencia, buscamos aprovechar las partes que se repiten entre clientes y empleados, como el nombre, teléfono y correo, para que nos permita organizar mejor el código y evitar escribir lo mismo varias veces, también, para facilitar futuras modificaciones.