UCSM Esc. Prof. de Ingeniería de Sistemas

INFORME DE PRÁCTICAS

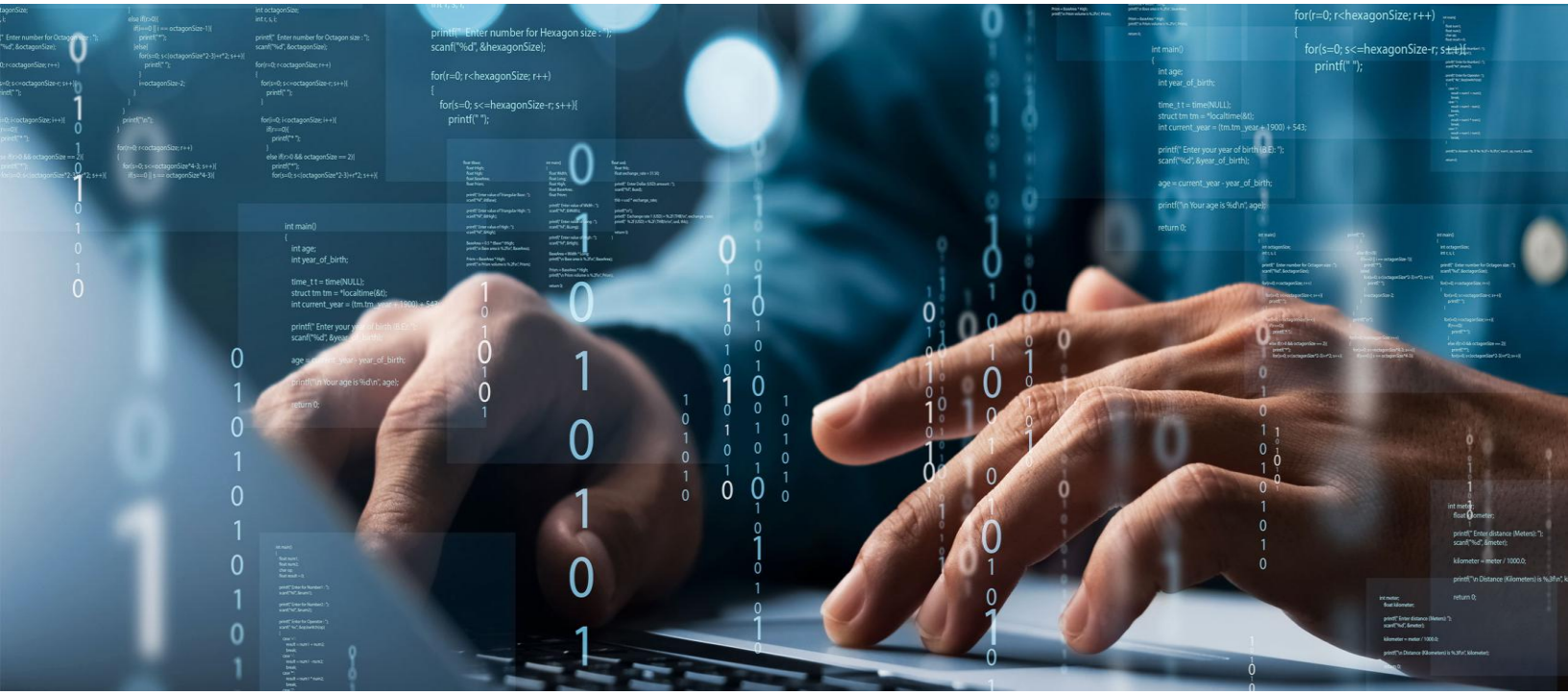
Marzo - 2025

LENGUAJES DE PROGRAMACIÓN II

Práctica N° 01: Introducción a la Programación Orientada a Objetos

Elaborado por:

YUJRA LAURA ALVARO DENNIS



GRUPO N° 07

PRÁCTICAS DE LENGUAJES DE PROGRAMACIÓN II

Presentado por:
74829039 YUJRA LAURA ALVARO DENNIS 100%

RECONOCIMIENTOS

Los autores de este trabajo reconocen con gratitud a los visionarios que dieron vida a Java. A James Gosling, el "padre de Java", y a su equipo en Sun Microsystems, quienes en la década de 1990 concibieron un lenguaje que revolucionaría la forma en que interactuamos con la tecnología. Su enfoque en la portabilidad, la seguridad y la facilidad de uso sentó las bases para un lenguaje que trascendería las plataformas y se convertiría en un estándar de la industria.

PALABRAS CLAVES

Clase, máquina virtual, paquete, POO.

ÍNDICE

ÍNDICE DE TABLAS Y FIGURAS

1. RESÚMEN

Este informe detalla el desarrollo de un sistema de gestión de inventario y pedidos en Python. El sistema permite registrar productos, generar órdenes de compra y emitir reportes de ventas y control de stock. Se emplean principios de Programación Orientada a Objetos (POO), implementando clases como `Producto`, `Pedido`, `AdministradorInventario` y `GeneradorDeReportes`.

2. INTRODUCCIÓN

El objetivo de esta práctica es modelar y desarrollar un sistema para la gestión de inventario y pedidos para una empresa de venta de productos electrónicos en línea. La implementación se realiza utilizando paradigmas de programación estructurada y POO, permitiendo una transición natural hacia un código más modular y reutilizable.

3. MARCO TEÓRICO

3.1 Escenario del Problema

La empresa requiere un sistema eficiente para manejar su inventario y los pedidos de sus clientes. El sistema debe permitir:

- Registrar productos con información detallada (nombre, precio, cantidad en inventario, etc.).
- Gestionar pedidos de clientes con múltiples productos y cantidades.
- Actualizar automáticamente el inventario tras la realización de un pedido.
- Generar reportes de ventas y análisis de inventario para la toma de decisiones.

3.1.1 Paradigmas de Programación

En esta práctica se exploran dos enfoques:

- **Programación Estructurada**: Uso de estructuras de datos y funciones independientes para gestionar productos y pedidos.
- **Programación Orientada a Objetos (POO)**: Implementación de clases como `Producto`, `Pedido`, `AdministradorInventario` y `GeneradorDeReportes`, mejorando la organización y escalabilidad del código.

Figura N° 1: código de función

...

4. EXPERIENCIAS DE PRÁCTICA

4.1 Expericia de Práctica N° 01: Paradigma de Programación Estructurada

Colocar código documentado

```
productos = []
pedidos = []

def registrar_producto(nombre, precio, cantidad):
    productos.append({"nombre": nombre, "precio": precio, "cantidad": cantidad})

def crear_pedido(nombre_producto, cantidad):
    pedidos.append({"producto": nombre_producto, "cantidad": cantidad})
```

4.2 Expericia de Práctica N° 02: Paradigma de POO con Clases Producto y Pedido

Colocar código documentado

```
class Producto:
    def __init__(self, nombre, precio, cantidad):
        self.nombre = nombre
        self.precio = precio
        self.cantidad = cantidad

    def obtener_detalle(self):
        return f"{self.nombre} - Precio: {self.precio} - Stock: {self.cantidad}"

class Pedido:
    def __init__(self):
        self.items = []

    def agregar_producto(self, producto, cantidad):
        self.items.append((producto, cantidad))
```

4.3 Expericia de Práctica N° 03: Implementación de AdministradorInventario

Colocar código documentado

```
class AdministradorInventario:
    def __init__(self):
        self.catalogo = {}

    def agregar_producto(self, producto):
        self.catalogo[producto.nombre] = producto

    def actualizar_stock(self, pedido):
        for producto, cantidad in pedido.items:
            if producto.nombre in self.catalogo:
                self.catalogo[producto.nombre].cantidad -= cantidad
```

4.4 Expericia de Práctica N° 04: Implementación de **GeneradorDeReportes

Colocar código documentado

```
class GeneradorDeReportes:
    def __init__(self, administrador):
        self.administrador = administrador

    def reporte_ventas(self):
        print(f"Total de pedidos procesados: {len(self.administrador.pedidos)}")

    def reporte_stock_bajo(self, limite=5):
        print("Productos con bajo stock:")
        for producto in self.administrador.catalogo.values():
```

```
            if producto.cantidad < limite:
                print(producto.obtener_detalle())
```

5. ACTIVIDADES

Colocar código documentado

```
class Producto:
    def __init__(self, codigo, nombre, precio, cantidad):
        self.codigo = codigo
        self.nombre = nombre
        self.precio = precio
        self.cantidad = cantidad

    def detalle(self):
        return f"Código: {self.codigo} | {self.nombre} | Precio: {self.precio}
| Stock: {self.cantidad}"


class Orden:
    def __init__(self, numero):
        self.numero = numero
        self.items = {}

    def agregar_item(self, producto_id, cantidad):
        if producto_id in self.items:
            self.items[producto_id] += cantidad
        else:
            self.items[producto_id] = cantidad

    def mostrar_orden(self):
        print(f"\n--- Orden #{self.numero} ---")
        for producto_id, cantidad in self.items.items():
            print(f"Producto {producto_id} - Cantidad: {cantidad}")


class AdministradorInventario:
    def __init__(self):
        self.catalogo = {}
        self.ordenes = []
        self.siguiente_codigo = 1
        self.siguiente_orden = 1

    def agregar_producto(self):
        nombre = input("Nombre del producto: ")
        precio = float(input("Precio: "))
        cantidad = int(input("Cantidad disponible: "))
        self.catalogo[self.siguiente_codigo] = Producto(self.siguiente_codigo,
nombre, precio, cantidad)
        print("Producto añadido correctamente.")
        self.siguiente_codigo += 1
```

```
    def generar_orden(self):
        nueva_orden = Orden(self.siguiente_orden)
        self.siguiente_orden += 1
        while True:
            try:
                prod_id = int(input("Código del producto: "))
                cant = int(input("Cantidad: "))
                if prod_id in self.catalogo and self.catalogo[prod_id].cantidad
>= cant:
                    nueva_orden.agregar_item(prod_id, cant)
                    self.catalogo[prod_id].cantidad -= cant
                else:
                    print("Stock insuficiente o producto no encontrado.")
            except ValueError:
                print("Entrada inválida. Intente nuevamente.")

            if input("¿Agregar otro producto? (s/n): ").lower() != 's':
                break
        self.ordenes.append(nueva_orden)
        print("Orden registrada con éxito.")

    def mostrar_catalogo(self):
        print("\n--- Inventario Disponible ---")
        for producto in self.catalogo.values():
            print(producto.detalle())


class Reportes:
    def __init__(self, admin):
        self.admin = admin

    def ventas_totales(self):
        print(f"\nTotal de órdenes procesadas: {len(self.admin.ordenes)}")

    def productos_bajo_stock(self, limite=5):
        print("\n--- Productos con Stock Bajo ---")
        for producto in self.admin.catalogo.values():
            if producto.cantidad < limite:
                print(producto.detalle())

    def reporte_completo(self):
        self.ventas_totales()
        self.productos_bajo_stock()


def ejecutar():
    admin = AdministradorInventario()
    reportes = Reportes(admin)

    opciones = {
        "1": admin.agregar_producto,
        "2": admin.generar_orden,
        "3": reportes.ventas_totales,
        "4": reportes.productos_bajo_stock,
        "5": reportes.reporte_completo,
        "6": admin.mostrar_catalogo
    }

    while True:
```

```
        print("\n===== MENÚ =====")
        print("1. Agregar Producto")
        print("2. Realizar Orden")
        print("3. Ver Reporte de Ventas")
        print("4. Productos con Bajo Stock")
        print("5. Reporte Completo")
        print("6. Mostrar Inventario")
        print("7. Salir")
        opcion = input("Seleccione una opción: ")

        if opcion == "7":
            print("Saliendo...")
            break

        accion = opciones.get(opcion)
        if accion:
            accion()
        else:
            print("Opción no válida, intente de nuevo.")


if __name__ == "__main__":
    ejecutar()
```

6. CONCLUSIONES DE LA PRÁCTICA:

- ✓ La transición de programación estructurada a POO facilita la escalabilidad y mantenimiento del código.
- ✓ La encapsulación y modularidad mejoran la organización y reusabilidad del sistema.
- ✓ La gestión del inventario mediante clases permite un mejor control de productos y pedidos.
- ✓ Los reportes automáticos brindan información clave para la toma de decisiones.

7. CUESTIONARIO

1. ¿Qué es un paradigma de programación?
   Un paradigma de programación es un enfoque que define la forma en que se estructura y organiza el código. Determina cómo los programas deben ser diseñados y escritos, basándose en principios y reglas específicas 【Horstmann, 2019】.

2. ¿Cuáles son los principales paradigmas de programación que existen?
   Los paradigmas principales son:

   Programación imperativa: Se basa en instrucciones secuenciales que modifican el estado del programa.
   Programación estructurada: Usa funciones y estructuras de control de flujo como if, while y for.
   Programación orientada a objetos (POO): Organiza el código en clases y objetos.
   Programación funcional: Usa funciones puras y evita el estado mutable.

Programación lógica: Se basa en reglas y hechos para resolver problemas 【Python Software Foundation, 2024】.

3. ¿En qué se enfoca la programación estructurada?
   Se enfoca en mejorar la legibilidad del código mediante el uso de estructuras de control bien definidas, como secuencias, decisiones y repeticiones. Evita el uso del goto y fomenta la modularidad mediante funciones 【Horstmann, 2019】.

4. ¿Qué son las estructuras de datos en la programación estructurada?
   Son formatos organizados para almacenar y manipular datos de manera eficiente. Ejemplos incluyen arreglos, listas enlazadas, pilas, colas y árboles 【Python Software Foundation, 2024】.

5. ¿En qué se enfoca la programación orientada a objetos?
   En encapsular datos y comportamientos dentro de objetos que interactúan entre sí. Promueve la reutilización del código mediante la herencia y mejora la organización a través de clases 【Horstmann, 2019】.

6. ¿Qué es una clase en la programación orientada a objetos?
   Es una plantilla que define los atributos y métodos de un conjunto de objetos similares. Actúa como un modelo a partir del cual se crean instancias 【Python Software Foundation, 2024】.

7. ¿Qué es un objeto en la programación orientada a objetos?
   Es una instancia de una clase que tiene atributos específicos y puede ejecutar métodos definidos en su clase 【Horstmann, 2019】.

8. ¿Cuál es la diferencia entre una clase y un objeto?
   Una clase es un modelo o plantilla, mientras que un objeto es una instancia concreta de esa clase con valores específicos en sus atributos 【Python Software Foundation, 2024】.

9. ¿Qué es la herencia en la programación orientada a objetos?
   Es un mecanismo que permite que una clase (subclase) adquiera atributos y métodos de otra clase (superclase). Facilita la reutilización del código y la especialización 【Horstmann, 2019】.

10. ¿Qué es el encapsulamiento en la programación orientada a objetos?
    Es el principio que restringe el acceso a los atributos y métodos internos de un objeto, exponiendo solo lo necesario a través de interfaces públicas 【Python Software Foundation, 2024】.

11. ¿Qué es el polimorfismo en la programación orientada a objetos?
    Es la capacidad de un objeto de comportarse de diferentes maneras según el contexto en el que se use. Se logra mediante sobrecarga de métodos y herencia 【Horstmann, 2019】.

12. ¿Cómo se pueden definir métodos y propiedades en una clase en la programación orientada a objetos?

Se definen dentro de una clase utilizando palabras clave como def en Python, public en Java o void en C++. Un método es una función dentro de una clase, y las propiedades son variables de la clase 【Python Software Foundation, 2024】.

13. ¿Qué es el constructor de una clase en la programación orientada a objetos?
    Es un método especial que se ejecuta automáticamente cuando se crea un objeto de la clase. Se usa para inicializar atributos. En Python, se define con __init__, en Java con el mismo nombre de la clase y en C++ con el mismo nombre de la clase también 【Horstmann, 2019】.
14. ¿Cómo se pueden crear y manipular objetos en la programación orientada a objetos?
    Se crean instancias de una clase usando su constructor. Luego, se manipulan mediante sus atributos y métodos, accediendo a ellos con objeto.metodo() o objeto.atributo según el lenguaje 【Python Software Foundation, 2024】.

15. ¿Qué es la abstracción en la programación orientada a objetos?
    Es el proceso de ocultar detalles internos de un objeto y exponer solo lo esencial. Facilita la interacción con objetos sin conocer su implementación interna 【Horstmann, 2019】.

16. ¿Qué ventajas ofrece la programación orientada a objetos respecto a la programación estructurada?
    - Modularidad y mejor organización del código.
    - Reutilización mediante herencia y polimorfismo.
    - Mayor facilidad de mantenimiento y escalabilida【Python Software Foundation, 2024】.
17. ¿Cómo se pueden combinar los paradigmas de programación estructurada y orientada a objetos para resolver problemas complejos?
    Se pueden utilizar estructuras de control dentro de métodos de clases en POO. También es común usar funciones estructuradas dentro de clases para resolver problemas específicos 【Horstmann, 2019】.

18. ¿Qué lenguajes de programación son orientados a objetos?
    Ejemplos incluyen Java, Python, C++, C#, Ruby y Swift 【Python Software Foundation, 2024】.

19. ¿Qué aplicaciones prácticas tienen los conceptos de la programación orientada a objetos?
    - Desarrollo de aplicaciones empresariales.
    - Creación de videojuegos.
    - Diseño de software de inteligencia artificial.
    - Desarrollo de aplicaciones móviles y web 【Horstmann, 2019】.

20. ¿Qué es un patrón de diseño en la programación orientada a objetos y cómo se puede utilizar para resolver problemas de diseño de software?
    Un patrón de diseño es una solución general a un problema recurrente en el desarrollo de software. Ejemplos incluyen Singleton (para garantizar una única instancia de una clase), Factory (para crear objetos de diferentes clases) y MVC (Modelo-Vista-Controlador, utilizado en aplicaciones web) 【Python Software Foundation, 2024】.

8. BIBLIOGRAFÍA

[1] Python Software Foundation. "Python Official Documentation." [Online]. Available: https://docs.python.org/3/

[2] C. Horstmann, "Big Java: Early Objects", John Wiley & Sons, 2019.