100 Ejercicios

Tema 3: Programación Orientada a Objetos (POO)

Temas Cubiertos:

1. Clases y Objetos
2. Herencia y Polimorfismo
3. Métodos y Atributos
4. Principios SOLID

1ra Parte: 50 Ejercicios para Practicar y Afianzar Conocimientos del Tema 3

Clases y Objetos

1. **Crear una Clase**: Define una clase Persona con atributos nombre y edad.
2. **Crear un Objeto**: Crea un objeto de la clase Persona e imprime sus atributos.
3. **Métodos**: Añade un método saludar() a la clase Persona que imprima un saludo utilizando el nombre de la persona.
4. **Modificar Atributos**: Escribe un programa que modifique el atributo edad de un objeto Persona.
5. **Constructor**: Añade un constructor a la clase Persona para inicializar los atributos nombre y edad.
6. **Encapsulamiento**: Utiliza encapsulamiento para hacer que los atributos de la clase Persona sean privados y proporciona métodos para acceder y modificar esos atributos.
7. **Herencia Simple**: Define una clase Empleado que herede de Persona y añada un atributo sueldo.
8. **Métodos Sobrescritos**: Sobrescribe el método saludar() en la clase Empleado para incluir información sobre el sueldo.

9. **Herencia Múltiple:** Define una clase Freelancer que herede de Persona y otra clase Proyecto, que contenga el atributo proyecto_actual.
10. **Métodos Especiales:** Añade el método __str__() a la clase Persona para que imprima una representación legible del objeto.

Herencia y Polimorfismo

11. **Polimorfismo Básico:** Crea una función que acepte un objeto de tipo Persona y llame al método saludar(), demostrando polimorfismo.
12. **Clases Abstractas:** Define una clase abstracta Figura con un método area() y crea subclases Circulo y Rectangulo que implementen el método area().
13. **Métodos Abstractos:** Utiliza el módulo abc para definir métodos abstractos en la clase Figura.
14. **Interface:** Define una interface Animal con un método sonido(), y crea clases Perro y Gato que implementen esta interface.
15. **Herencia en Cadena:** Define una clase Vehiculo, luego una clase Automovil que herede de Vehiculo y finalmente una clase Electrico que herede de Automovil.
16. **Super():** Utiliza la función super() para llamar al constructor de la clase base desde la clase derivada.
17. **Polimorfismo con Listas:** Crea una lista de objetos de diferentes clases que comparten un método en común y llama a ese método en un bucle.
18. **Herencia de Métodos Privados:** Demuestra cómo los métodos privados no son accesibles desde las subclases.
19. **Herencia con Atributos Privados:** Muestra cómo los atributos privados de una clase base pueden ser utilizados en una subclase.
20. **Mixins:** Crea una clase Mixin con métodos que pueden ser utilizados por otras clases sin heredar de Mixin.

Métodos y Atributos

21. **Atributos de Clase:** Define un atributo de clase contador en Persona que lleve la cuenta de cuántos objetos Persona se han creado.

22. **Atributos de Instancia:** Diferencia entre atributos de instancia y de clase mediante un ejemplo práctico.

23. **Atributos Dinámicos:** Añade atributos a un objeto Persona de forma dinámica después de su creación.

24. **Métodos de Clase:** Define un método de clase obtener_contador() en Persona que devuelva el valor del contador.

25. **Métodos Estáticos:** Define un método estático es_mayor_de_edad(edad) que devuelva True si la edad es mayor o igual a 18.

26. **Propiedades:** Utiliza propiedades (@property) para acceder y modificar atributos privados.

27. **Métodos Mágicos:** Implementa métodos mágicos como __add__ para sumar dos objetos Empleado y devolver la suma de sus sueldos.

28. **Métodos de Comparación:** Implementa métodos de comparación (__eq__, __lt__, etc.) para comparar objetos Persona por su edad.

29. **Métodos Encadenados:** Implementa métodos encadenados en una clase Cadena, de manera que se puedan llamar varios métodos en una sola línea.

30. **Atributos Calculados:** Define un atributo calculado IMC (Índice de Masa Corporal) en una clase Persona que se calcule a partir de los atributos peso y altura.

Principios SOLID

31. **Responsabilidad Única:** Escribe una clase que cumpla con el principio de responsabilidad única.

32. **Abierto/Cerrado:** Escribe una clase que cumpla con el principio de abierto/cerrado y demuestra cómo se puede extender sin modificar.

33. **Sustitución de Liskov:** Demuestra el principio de sustitución de Liskov mediante un ejemplo de clases base y derivadas.

34. **Segregación de Interfaces:** Escribe varias interfaces pequeñas en lugar de una interfaz grande para demostrar la segregación de interfaces.

35. **Inversión de Dependencias:** Implementa un ejemplo que cumpla con el principio de inversión de dependencias utilizando inyección de dependencias.

Ejercicios de Repaso

36. **Biblioteca:** Crea un sistema de biblioteca con clases Libro, Autor y Usuario, implementando métodos para prestar y devolver libros.

37. **Sistema de Reservas:** Desarrolla un sistema de reservas de vuelos con clases Vuelo, Pasajero y Reserva.

38. **Inventario:** Crea un sistema de inventario con clases Producto, Categoria y Proveedor, implementando métodos para añadir y quitar productos.

39. **Sistema Bancario:** Desarrolla un sistema bancario con clases Cuenta, Cliente y Transaccion.

40. **Juego de Cartas:** Crea un juego de cartas básico con clases Carta, Mazo y Jugador.

41. **Sistema de Gestión Escolar:** Desarrolla un sistema de gestión escolar con clases Estudiante, Profesor y Curso.

42. **Sistema de Gestión de Proyectos:** Implementa un sistema de gestión de proyectos con clases Proyecto, Tarea y Empleado.

43. **Sistema de Ventas:** Crea un sistema de ventas con clases Venta, Cliente y Producto.

44. **Sistema de Gestión de Hospitales:** Desarrolla un sistema de gestión de hospitales con clases Paciente, Doctor y Cita.

45. **Simulación de Tráfico:** Implementa una simulación de tráfico con clases Vehiculo, Carretera y Semaforo.

Desafíos de Código

46. **Simulación de Ecosistema**: Crea una simulación básica de un ecosistema con clases Animal, Planta y Ecosistema.
47. **Red Social**: Desarrolla una simulación de red social con clases Usuario, Publicacion y Comentario.
48. **Sistema de Votación**: Implementa un sistema de votación con clases Votante, Candidato y Eleccion.
49. **Sistema de Tickets**: Desarrolla un sistema de tickets con clases Ticket, Cliente y Agente.
50. **Simulación de Mercado**: Crea una simulación de mercado con clases Vendedor, Comprador y Producto.

2da Parte: 50 Ejercicios Avanzados sobre el Tema 3

Clases y Objetos

1. **Metaclases**: Define una metaclase que modifique dinámicamente los atributos de las clases que crea.
2. **Singleton**: Implementa el patrón Singleton en Python.
3. **Proxy**: Implementa el patrón Proxy para controlar el acceso a un objeto.
4. **Decorador de Clases**: Escribe un decorador de clases que añada nuevos métodos a una clase.
5. **Clase Fábrica**: Implementa una clase fábrica que cree instancias de diferentes clases basadas en una entrada.
6. **Builder**: Implementa el patrón Builder para construir objetos complejos paso a paso.
7. **Prototype**: Implementa el patrón Prototype para clonar objetos.
8. **Adaptador**: Implementa el patrón Adaptador para que dos clases incompatibles puedan trabajar juntas.
9. **Memento**: Implementa el patrón Memento para guardar y restaurar el estado de un objeto.

10. **Comando:** Implementa el patrón Comando para encapsular una solicitud como un objeto.

Herencia y Polimorfismo

11. **Visitor:** Implementa el patrón Visitor para agregar nuevas operaciones a clases sin cambiar su código.
12. **Chain of Responsibility:** Implementa el patrón Chain of Responsibility para pasar una solicitud a lo largo de una cadena de manejadores.
13. **Interpreter:** Implementa el patrón Interpreter para interpretar un lenguaje simple.
14. **State:** Implementa el patrón State para cambiar el comportamiento de un objeto cuando cambia su estado interno.
15. **Strategy:** Implementa el patrón Strategy para definir una familia de algoritmos, encapsular cada uno y hacerlos intercambiables.
16. **Template Method:** Implementa el patrón Template Method para definir el esqueleto de un algoritmo en una operación y dejar que las subclases redefinan ciertos pasos del algoritmo.
17. **Composite:** Implementa el patrón Composite para tratar objetos individuales y compuestos de manera uniforme.
18. **Flyweight:** Implementa el patrón Flyweight para usar el menor número posible de objetos compartiendo tantos datos como sea posible.
19. **Bridge:** Implementa el patrón Bridge para desacoplar una abstracción de su implementación.
20. **Mediator:** Implementa el patrón Mediator para reducir las dependencias entre objetos comunicándolos a través de un mediador.

Métodos y Atributos

21. **Descriptores:** Implementa descriptores para controlar el acceso a los atributos de una clase.

22. **Slots:** Utiliza __slots__ para limitar los atributos que se pueden añadir a una clase.
23. **Decoradores con Argumentos:** Implementa un decorador que acepte argumentos y modifique el comportamiento de un método.
24. **Context Managers:** Implementa un administrador de contexto personalizado utilizando la clase contextlib.ContextDecorator.
25. **Metaprogramación:** Utiliza type() para crear dinámicamente una nueva clase.
26. **Enumeraciones:** Utiliza el módulo enum para definir y utilizar enumeraciones en una clase.
27. **Operadores Sobrecargados:** Sobrecarga operadores aritméticos (+, -, *, /) en una clase personalizada.
28. **Propiedades Avanzadas:** Define propiedades que realicen validaciones y conversión de tipos al establecerse.
29. **Descriptores con Cache:** Implementa un descriptor que cachee los resultados de una propiedad costosa de calcular.
30. **Atributos de Solo Lectura:** Utiliza propiedades para definir atributos de solo lectura en una clase.

Principios SOLID

31. **Refactorización con SOLID:** Refactoriza una clase para que cumpla con todos los principios SOLID.
32. **Aplicación de SRP:** Refactoriza una clase con múltiples responsabilidades para que cumpla con el principio de responsabilidad única (SRP).
33. **Aplicación de OCP:** Extiende una clase sin modificar su código original para que cumpla con el principio abierto/cerrado (OCP).
34. **Aplicación de LSP:** Reemplaza una clase base por una subclase sin romper el programa para cumplir con el principio de sustitución de Liskov (LSP).

35. **Aplicación de ISP:** Divide una interfaz grande en interfaces más pequeñas y específicas para cumplir con el principio de segregación de interfaces (ISP).

36. **Aplicación de DIP:** Refactoriza una clase para que dependa de abstracciones en lugar de implementaciones concretas, cumpliendo con el principio de inversión de dependencias (DIP).

Ejercicios de Repaso

37. **Simulación de Ecosistema Avanzada:** Mejora la simulación del ecosistema para incluir más tipos de organismos y relaciones complejas.

38. **Sistema de Gestión Escolar Avanzado:** Añade funcionalidades avanzadas como horarios de clases y calificaciones al sistema de gestión escolar.

39. **Juego de Cartas Avanzado:** Implementa reglas más complejas y modos de juego adicionales al juego de cartas básico.

40. **Sistema de Reservas Avanzado:** Mejora el sistema de reservas de vuelos para incluir diferentes clases de boletos y políticas de cancelación.

41. **Sistema Bancario Avanzado:** Añade funcionalidades como transferencias internacionales y préstamos al sistema bancario.

42. **Sistema de Gestión de Proyectos Avanzado:** Mejora el sistema de gestión de proyectos para incluir gráficos de Gantt y reportes de progreso.

43. **Sistema de Ventas Avanzado:** Añade funcionalidades como descuentos y promociones al sistema de ventas.

44. **Sistema de Gestión de Hospitales Avanzado:** Mejora el sistema de gestión de hospitales para incluir historial médico y recetas electrónicas.

45. **Simulación de Tráfico Avanzada:** Añade funcionalidades como diferentes tipos de vehículos y condiciones de tráfico a la simulación de tráfico.

46. **Sistema de Biblioteca Avanzado:** Añade funcionalidades como reservas de libros y recomendaciones al sistema de biblioteca.

Desafíos de Código

47. **Simulación de Ecosistema Completa:** Implementa una simulación completa de un ecosistema con múltiples especies y relaciones tróficas.

48. **Red Social Completa:** Desarrolla una red social completa con funcionalidades como mensajes privados y grupos.

49. **Sistema de Votación Completo:** Implementa un sistema de votación completo con diferentes tipos de elecciones y resultados en tiempo real.

50. **Sistema de Tickets Completo:** Desarrolla un sistema de tickets completo con funcionalidades como prioridades y seguimiento de tiempo.