Análisis de Caso

Paradigma de orientación a objetos👨‍💻

Análisis de Caso

Paradigma de orientación a objetos

📍 Situación inicial

La empresa **TechSolutions** se especializa en el desarrollo de software para pequeñas y medianas empresas. Recientemente, han identificado problemas en la escalabilidad y mantenimiento de su sistema de gestión de clientes, ya que fue desarrollado utilizando **programación estructurada**. Como resultado, modificar el código existente es complejo y propenso a errores.

El equipo de desarrollo ha decidido migrar a un enfoque basado en **Programación Orientada a Objetos (POO)**, lo que permitirá mejorar la modularidad, reutilización de código y mantenimiento del sistema. Sin embargo, algunos miembros del equipo no están familiarizados con este paradigma y necesitan comprender sus fundamentos antes de iniciar la reestructuración del software.

🔎Descripción del Caso

Como **consultor de desarrollo de software**, tu misión es ayudar al equipo de **TechSolutions** a comprender y aplicar los principios de **Programación Orientada a Objetos (POO)**. Para ello, deberás:

1. **Explicar los conceptos fundamentales de la POO** y cómo se diferencian de la programación estructurada.
2. **Definir clases y objetos en Python**, destacando cómo estos permiten una mejor organización del código.
3. **Implementar un modelo de gestión de clientes basado en POO**, utilizando clases, atributos y métodos.
4. **Mostrar cómo la encapsulación y abstracción pueden mejorar la seguridad y claridad del código**.

El objetivo es que el equipo pueda aplicar estos conocimientos para rediseñar su software utilizando POO.

💡 Instrucciones

Para cumplir con esta misión, sigue los siguientes pasos:

Paso 1: Comparación entre Programación Estructurada y POO

- Explica las diferencias entre ambos paradigmas.
- Muestra un ejemplo en Python de una implementación en **programación estructurada** y su equivalente en **POO**.

Paso 2: Creación de una Clase Cliente

- Define una clase Cliente que tenga atributos como nombre, email, saldo.
- Implementa métodos para **registrar un cliente**, **actualizar sus datos** y **consultar su saldo**.
- Usa el método __init__ para inicializar los objetos correctamente.

Paso 3: Aplicación de Encapsulamiento y Abstracción

- Protege los atributos sensibles con modificadores de acceso (_ o __).
- Implementa métodos públicos que permitan acceder y modificar la información de manera controlada.

Paso 4: Interacción entre Objetos

- Crea una segunda clase Banco, que gestione múltiples clientes.
- Agrega métodos en la clase Banco para **agregar clientes**, **realizar transferencias** y **consultar información de clientes**.

Paso 5: Implementación del Modelo en Código

- Escribe el código Python necesario para modelar estas clases.
- Implementa un ejemplo práctico donde se creen clientes, se almacenen en un banco y se realicen operaciones.

📬Entregables

Al finalizar el análisis, se deben presentar los siguientes entregables:

1. **Explicación Comparativa** 📄
   - Documento con la explicación de las diferencias entre **Programación Estructurada y POO**, con ejemplos en Python.
2. **Código de la Clase Cliente** 📝
   - Implementación en Python de la clase Cliente con sus atributos y métodos.
3. **Código de la Clase Banco** 🏦
   - Implementación en Python de la clase Banco, mostrando cómo gestiona múltiples clientes.
4. **Ejemplo Práctico** 🔄
   - Código de una simulación donde se creen varios clientes y se realicen operaciones básicas con ellos.

¡Éxitos!

Nos vemos más adelante

alkemy