**Informe de Trabajo Práctico 3: Objeto, Clase, UML y Java: un enfoque práctico**

**1. Objetos y Clases**

Uno de los primeros conceptos que aprendimos en POO es el de **objetos** y **clases**. Los objetos representan entidades del mundo real, y las clases son plantillas que definen sus atributos y comportamientos (métodos). Por ejemplo, en uno de nuestros ejercicios, creamos la clase Producto para gestionar productos de una farmacia. Los atributos como codigo, descripcion, y costo representan las características del producto, mientras que los métodos como precioLista() o ajustarPtoRepo() nos permitieron manejar el stock y los precios de manera dinámica.

Aquí aprendimos la importancia de la **encapsulación**, que asegura que los datos de un objeto estén protegidos y solo puedan ser modificados mediante métodos controlados (como los **getters** y **setters**). Esto evita errores al modificar los atributos directamente y mejora la seguridad del sistema.

**2. Encapsulamiento y Ocultación de Información**

El **encapsulamiento** se relaciona directamente con la **ocultación de información**, que limita el acceso directo a los datos de un objeto. Un claro ejemplo lo vimos al implementar el sistema bancario con la clase CuentaCorriente. Los atributos como saldo estaban encapsulados, y sólo métodos como depositar() o extraer() podían modificarlos​. De esta manera, el saldo no podía ser alterado por accidente desde fuera de la clase, manteniendo la integridad de la cuenta.

**3. Herencia y Polimorfismo**

Otra parte importante de POO que aprendimos es la **herencia**, que nos permite reutilizar código. Vimos este concepto al trabajar con figuras geométricas como círculos y rectángulos. Ambas figuras heredaban de una clase base llamada Figura, que contenía características comunes, pero cada figura implementaba sus propios cálculos para el área o perímetro según su forma​.

El **polimorfismo** fue clave en este ejercicio, ya que aunque ambos objetos (círculo y rectángulo) respondían al mismo método (superficie()), el cálculo era diferente para cada uno. Este concepto nos permitió crear código más flexible, ya que una misma llamada a un método generaba resultados distintos según la clase del objeto.

**4. Abstracción y Modelado del Mundo Real**

La **abstracción** nos ayudó a simplificar problemas complejos del mundo real. En el sistema de gestión hospitalaria que desarrollamos, usamos la abstracción al crear clases como Paciente, Hospital, y Localidad​. Cada una de estas clases representaba un elemento esencial del sistema, permitiéndonos enfocarnos en los detalles importantes para resolver el problema sin distraernos con aspectos irrelevantes.

La abstracción también nos permitió mejorar y ampliar el sistema de manera sencilla. Si en el futuro quisiéramos agregar más atributos o comportamientos a la clase Paciente, podríamos hacerlo sin necesidad de rediseñar todo el sistema, lo que mejora la mantenibilidad del código.

**Conclusión**

La Programación Orientada a Objetos nos ha permitido aprender a resolver problemas complejos mediante el uso de objetos que interactúan entre sí. La encapsulación, herencia, polimorfismo y abstracción nos ayudan a crear sistemas modulares, reutilizables y fáciles de mantener. Estos conceptos no solo son esenciales para construir software robusto, sino que también las buenas prácticas de documentación y entrega fomentan una mentalidad profesional desde el principio. Estas lecciones son fundamentales para nuestra carrera y nos han preparado para enfrentar desafíos mayores en el futuro.