Ejercicios Propuestos - POO

Ejercicio 1: Creación y definición de una clase.

Definir una clase de nombre **Persona**: La responsabilidad de la clase será modelar los datos de una persona.

La clase **Persona** posee nombre, apellido, número de documento de identidad y año de nacimiento. La clase debe tener un constructor que inicialice los valores de sus respectivos atributos y debe incluir un método que permita mostrar en pantalla los valores de los atributos del objeto.

En main se deben instanciar dos personas y mostrar los valores de sus atributos en pantalla.

Agregar dos nuevos atributos a la clase Persona. Un atributo que represente el país de nacimiento de la persona (de tipo String) y otro que identifique el género de la persona, el cual debe representarse como un char con valores 'H' o 'M'.

Modificar el constructor de la clase Persona para que inicialice estos dos nuevos atributos.

Modificar el método imprimir de la clase Persona para que muestre en pantalla los valores de los nuevos atributos.

Ejercicio 2: Sistema Solar

Se requiere un programa que modele el concepto de un planeta del sistema solar. Definir una clase **Planeta** con los siguientes atributos:

- ★ Un nombre de tipo **String** con valor inicial de **null.**
- ★ Cantidad de satélites **int** con valor inicial de **cero.**
- ★ Masa en kilogramos **float** con valor inicial de **cero.**
- ★ Volumen en kilómetros cúbicos **float** con valor inicial de **cero.**
- ★ Diámetro en kilómetros de tipo **int** con valor inicial de **cero.**
- ★ Distancia media al Sol en **millones de kilómetros**, de tipo **int** con valor inicial de **cero.**
- ★ Un atributo observable de tipo **booleano** con valor inicial **False** para determinar si es observable o no.

El atributo para identificar el tipo de planeta se expresa como una asociación entre la clase **Planeta** y la clase **TipoPlaneta**. Tipo de planeta posee los siguientes valores **GASEOSO, TERRESTRE y ENANO.**

La clase debe incluir los siguientes métodos:

- ➢ Un constructor que inicialice los valores de sus respectivos atributos.
- ➢ Un método que imprima en pantalla los valores de los atributos de un planeta.
- ➢ **NOTA:** Utilizar las propiedades de los atributos de getters y setters para permitir el acceso y la modificación de los valores de manera controlada.
- ➢ Calcular la densidad de un planeta, como el cociente entre su masa y su volumen.
- ➢ Para determinar si un planeta es observable o no, el ángulo aparente (en grados) debe ser mayor a 20°. Dicho cálculo se realiza mediante la siguiente expresión, cuyo resultado se obtiene en radianes:

$$\theta = 2 \cdot arctan\left(\frac{Diámetro}{2 \cdot Distancia\ media\ al\ Sol}\right)$$

IMPORTANTE: Utilizar la librería **math** de python para realizar el cálculo y convertir de radianes a grados. En caso de ser observable se debe modificar el atributo **Observable** por **True.**

- ➢ Agregar dos atributos a la clase Planeta. El primero debe representar el periodo **orbital del planeta (en años).** El segundo atributo representa el **periodo de rotación (en días)**.
- ➢ Modificar el constructor de la clase para que inicialice los valores de estos dos nuevos atributos.
- ➢ Modificar el método imprimir para que muestre en pantalla los valores de los nuevos atributos.
- ➢ En main se deben crear dos planetas y mostrar los valores de sus atributos en pantalla. Además, se debe imprimir la densidad de cada planeta y si el planeta es un planeta exterior del sistema solar.

Ejercicio 3: Automóvil

Se requiere un programa que modele el concepto de un automóvil.

Un automóvil tiene los siguientes atributos:

- ❖ **Marca:** El nombre del fabricante.
- ❖ **Modelo:** Año de fabricación.
- ❖ **Motor:** Volumen en litros por cilindrada del motor de un automóvil.
- ❖ **Tipo de combustible:** Valor enumerado con los posibles valores de gasolina, bioetanol, diésel, biodiésel, gas natural.
- ❖ **Tipo de automóvil:** Valor enumerado con los posibles valores de carro de ciudad, subcompacto, compacto, familiar, ejecutivo, SUV.
- ❖ **Número de puertas:** Cantidad de puertas.

- **Cantidad de asientos:** Número de asientos disponibles que tiene el vehículo.
- **Velocidad máxima:** Velocidad máxima sostenida por el vehículo en km/h.
- **Color:** Valor enumerado con los posibles valores de blanco, negro, rojo, naranja, amarillo, verde, azul, violeta.
- **Velocidad actual:** velocidad del vehículo en un momento dado.

La clase debe incluir los siguientes métodos:

- Un constructor para la clase Automóvil donde se le pasen como parámetros los valores de sus atributos.
- Métodos **getter** y **setter** para la clase Automóvil.
- Métodos para acelerar una cierta velocidad, desacelerar y frenar (colocar la velocidad actual en cero). Es importante tener en cuenta que no se debe acelerar más allá de la velocidad máxima permitida para el automóvil. De igual manera, tampoco es posible desacelerar a una velocidad negativa. Si se cumplen estos casos, se debe mostrar por pantalla los mensajes correspondientes.
- Un método para calcular el tiempo estimado de llegada, utilizando como parámetro la distancia a recorrer en kilómetros. El tiempo estimado se calcula como el cociente entre la distancia a recorrer y la velocidad actual.
- Un método para mostrar los valores de los atributos de un automóvil en pantalla.
- En main se debe instanciar un automóvil, colocar su velocidad actual en 100 km/h, aumentar su velocidad en 20 km/h, luego decrementar su velocidad en 50 km/h, y después frenar. Con cada cambio de velocidad, se debe mostrar en pantalla la velocidad actual.
- Agregar a la clase Automóvil, un atributo para determinar si el vehículo es automático o no. Agregar los métodos getter y setter para dicho atributo. Modificar el constructor para inicializar dicho atributo.
- Modificar el método **acelerar** para que si la velocidad máxima se sobrepase se genere una multa. Dicha multa se puede incrementar cada vez que el vehículo intenta superar la velocidad máxima permitida.
- Agregar un método para determinar si un vehículo tiene multas y otro método para determinar el valor total de multas de un vehículo.

Ejercicio 4: Figuras Geométricas

Se requiere un programa que modele varias figuras geométricas: el **círculo**, el **rectángulo**, el **cuadrado** y el **triángulo rectángulo**.

- ★ El círculo tiene como atributo su radio en centímetros.
- ★ El rectángulo, su base y altura en centímetros.

- ★ El cuadrado, la longitud de sus lados en centímetros.
- ★ El triángulo, su base y altura en centímetros.

Se requieren métodos para determinar el área y el perímetro de cada fígura geométrica. Además, para el triángulo rectángulo se requiere:

- ❖ Un método que calcule la hipotenusa del rectángulo.
- ❖ Un método para determinar qué tipo de triángulo es:
- ❖ **Equilátero:** todos sus lados son iguales.
- ❖ **Isósceles:** tiene dos lados iguales.
- ❖ **Escaleno:** todos sus lados son diferentes.
- ❖ En main crear las cuatro figuras y probar los métodos respectivos.
- ❖ Agregar una nueva clase denominada **Rombo**. Definir los métodos para calcular el área y el perímetro de esta nueva figura geométrica.
- ❖ Agregar una nueva clase denominada **Trapecio**. Definir los métodos para calcular el área y el perímetro de esta nueva figura geométrica.

Ejercicio 5: Cuenta Bancaria

Se requiere un programa que modele una cuenta bancaria que posee los siguientes atributos:

- ❖ Nombres del titular.
- ❖ Apellidos del titular.
- ❖ Número de la cuenta bancaria.
- ❖ **Tipo de cuenta:** puede ser una cuenta de ahorros o una cuenta corriente.
- ❖ Saldo de la cuenta.
- ❖ Se debe definir un constructor que inicialice los atributos de la clase.
- ❖ Cuando se crea una cuenta bancaria, su saldo inicial tiene un valor de cero.

<u>En una determinada cuenta bancaria se puede:</u>

- ❖ Imprimir por pantalla los valores de los atributos de una cuenta bancaria.
- ❖ Consultar el saldo de una cuenta bancaria.
- ❖ Consignar un determinado valor en la cuenta bancaria, actualizando el saldo correspondiente.
- ❖ Retirar un determinado valor de la cuenta bancaria, actualizando el saldo correspondiente. Es necesario tener en cuenta que no se puede realizar el retiro si el valor solicitado supera el saldo actual de la cuenta.
- ❖ Agregar a la clase **CuentaBancaria**, un atributo que represente el porcentaje de interés mensual aplicado a la cuenta.

- ❖ Agregar un método que calcule un nuevo saldo aplicando la tasa de interés correspondiente.
- ❖ Se requiere modificar el programa de la cuenta bancaria para que realice las siguientes actividades:
- ❖ Comparar saldos entre cuentas bancarias. La cuenta para comparar es un objeto que se envía como parámetro del método. El método devuelve un valor booleano de verdadero si la cuenta actual es mayor o igual a la cuenta que se pasó como parámetro.
- ❖ Transferir dinero de una cuenta bancaria a otra. El método debe recibir como parámetro la cuenta de destino y el valor a transferir. El saldo de la cuenta actual debe disminuir el valor a transferir y el saldo de la cuenta destino debe aumentar. El método debe reutilizar el método retirar para evaluar si la cantidad a transferir se encuentra en la cuenta de origen.

Ejercicio N°6: Cafetería

Consigna General:

Desarrollar un sistema para una cadena de cafeterías que tiene varias sucursales distribuidas en una ciudad. El sistema debe modelar aspectos como el inventario, el personal de la sucursal, la relación con un proveedor, y la interacción con los clientes. El desarrollo se realizará en etapas para comprender gradualmente los conceptos de Programación Orientada a Objetos (POO).

Etapas del Proyecto

Etapa 1: Modelar Productos y Menú (Nivel Básico)

Objetivo: Introducir el concepto de clases y objetos.

Tarea: Crear una clase **Producto** que represente los productos que vende la cafetería.

- ❖ Cada producto debe tener un nombre y un precio.
- ❖ Crear una clase **Menú** (carta) que contenga una lista de productos.
- ❖ Implementar un método que permita agregar productos al menú.
- ❖ Crear un menú de ejemplo con al menos 3 productos y mostrarlos en pantalla.
- ❖ Definir el método mágico str para que un producto muestre su descripción

Etapa 2: Sucursales (Nivel Intermedio)

Objetivo: Introducir la relación de agregación y composición.

Tarea: Crear una clase **Sucursal**, la cual debe tener una ubicación y un menú propio.

- La sucursal debe poder mostrar su menú.
- Crear al menos dos sucursales y agregar productos distintos a cada una.
- Mostrar los menús de ambas sucursales en pantalla.

Etapa 3: Personal de la Cafetería (Nivel Intermedio)

Objetivo: Introducir el uso de clases para modelar roles.

Tarea: Crear clases para los empleados:

- **Cajero:** Encargado de cobrar a los clientes.
- **Barista:** Encargado de preparar los pedidos.
- Asignar al menos un cajero y un barista a cada sucursal.
- Implementar métodos para mostrar la información del personal de cada sucursal.

Etapa 4: Proveedor y Gestión de Stock (Nivel Avanzado)

Objetivo: Introducir el concepto de interacción entre clases.

Tarea: Crear una clase **Proveedor** que abastezca a todas las sucursales.

- Crear una clase **Encargado de Inventario** (stock manager) para cada sucursal. Este empleado es responsable de verificar el stock y hacer pedidos al proveedor.
- El encargado debe verificar si hay stock suficiente de un producto. Si no lo hay, se realiza un pedido al proveedor.
- Cuando el proveedor entrega el producto, el encargado lo recibe y lo agrega al inventario.
- Implementar métodos para verificar el stock y realizar pedidos.

Etapa 5: Clientes y Pedidos (Nivel Avanzado)

Objetivo: Modelar la interacción con los clientes.

Tarea: Los clientes deben poder pedir productos al cajero.

- El cajero toma el pedido, y si hay stock, el barista lo prepara.

- ❖ Modelar la opción de que el cliente pueda tomar el café en la sucursal o llevarlo.
- ❖ Implementar la funcionalidad para cobrar el pedido y descontar el stock.

Etapa 6: Control de Precios y Promociones (Nivel Avanzado)

Objetivo: Completar la funcionalidad con estrategias de precios y promociones.

Tarea: Crear una clase **Gerente de Precios**, que sea responsable de establecer los precios de los productos y definir un combo especial diferente para cada día de la semana.

- ❖ Implementar un método para mostrar el combo del día.
- ❖ Asegurar que los precios de los productos puedan ser ajustados dinámicamente por el gerente.

Producto Final:

Al finalizar todas las etapas, se habrá desarrollado un sistema completo que modela:

- ❖ Varias sucursales de una cadena de cafeterías.
- ❖ Un menú con diferentes productos en cada sucursal.
- ❖ Personal responsable de distintas tareas (cajero, barista, encargado de inventario).
- ❖ Gestión del stock y la interacción con un proveedor centralizado.
- ❖ Control de precios y promociones por parte de un gerente.

Ejercicio 7: Tienda de computación

Se requiere desarrollar un programa que modele una tienda de computadoras.

La clase Tienda posee los siguientes atributos:

Nombre de la tienda.

Propietario de la tienda.

Identificador de la tienda.

Cantidad de computadoras

Las Computadoras de la Tienda tienen los siguientes atributos:

· Marca de la computadora.

· Cantidad de memoria.

· Características del procesador.

· Sistema operativo.

· Precio de la computadora

El programa debe poseer métodos que permitan:

· **Agregar** una computadora a la tienda.

· **Eliminar** una computadora de la tienda dada su marca.

· **Buscar** una computadora en la tienda dada su marca.

· **Listar** la información de todas las computadoras que tiene la tienda.

Una Tienda posee muchas computadoras; esta relación se modela en UML mediante una asociación que conecta las clases Tienda y Computadora.

La clase Tienda tiene los atributos privados. Tiene un constructor y métodos públicos para determinar si la tienda está llena (tiendaLlena)o vacía (tiendaVacía), para añadir; buscar y eliminar computadores en la tienda y para imprimir los datos de la tienda y todos sus computadoras en pantalla.

La clase Computadora tiene los atributos privados. Tiene un constructor y no tiene métodos adicionales.

Ejercicio 8: Cuenta bancaria

Desarrollar un programa que modele una cuenta bancaria que tiene los siguientes atributos, que deben ser de acceso protegido:

- Saldo, de tipo float.
- Cantidad de depósitos con valor inicial cero, de tipo int.
- Cantidad de retiros con valor inicial cero, de tipo int.
- Tasa anual (porcentaje), de tipo float.
- Comisión mensual con valor inicial cero, de tipo float.

La clase **Cuenta** tiene un constructor que inicializa los atributos saldo y tasa anual con valores pasados como parámetros. La clase Cuenta tiene los siguientes métodos:

- **Depositar** una cantidad de dinero en la cuenta actualizando su saldo (+).
- **Retirar** una cantidad de dinero en la cuenta actualizando su saldo (-).

El valor a retirar no debe superar el saldo.

- Calcular el interés mensual de la cuenta y actualizar el saldo correspondiente.
- **Extracto mensual:** actualiza el saldo restándole la comisión mensual y calculando el interés mensual correspondiente (invoca el método anterior).
- **Imprimir:** muestra en pantalla los valores de los atributos.

Herencia: la clase **Cuenta** tiene dos clases hijas:

Caja de ahorros: posee un atributo para determinar si la Caja de ahorros está activa (tipo bool). Si el saldo es menor a $10000, la cuenta está inactiva, en caso contrario se considera activa. Los siguientes métodos se redefinen:

- **Depositar:** se puede depositar dinero si la cuenta está activa. Debe invocar al método heredado.
- **Retirar:** es posible retirar dinero si la cuenta está activa. Debe invocar al método heredado.
- **Extracto mensual:** si el número de retiros es mayor que 4, por cada retiro adicional, se cobra $1000 como comisión mensual.
- Al generar el extracto, se determina si la cuenta está activa o no con el saldo.
- Un método **imprimir** que muestre en pantalla el saldo de la cuenta, la comisión mensual y el número de transacciones realizadas (suma de cantidad de depositos y retiros).

Cuenta corriente: posee un atributo de sobregiro, el cual se inicializa en cero. Se redefinen los siguientes métodos:

- **Retirar:** se retira dinero de la cuenta actualizando su saldo. Se puede retirar dinero superior al saldo. El dinero que se debe queda como sobregiro.
- **Depositar:** invoca al método heredado. Si hay sobregiro, la cantidad consignada reduce el sobregiro.
- **Extracto mensual:** invoca al método heredado.
- Un nuevo método que muestra en pantalla el saldo de la cuenta, la comisión mensual, el número de transacciones realizadas (suma de cantidad de consignaciones y retiros) y el valor de sobregiro.
- En main implementar un objeto Caja de ahorros que llame a los métodos correspondientes.