Actividad Semanal 01 - Programación Orientada a Objetos, Abstracción y Encapsulamiento

Fecha de entrega: viernes 28 de febrero antes de las 23:59 hrs.

Objetivo: Aplicar los principios de abstracción y encapsulamiento en la resolución de problemas utilizando Java.

Instrucciones: Implementa cada uno de los siguientes problemas asegurándote de cumplir con los requisitos indicados. No se permite el uso de herencia ni polimorfismo.

1. **Gestión de Cuentas Bancarias**

Descripción: Implementa una clase **Cuenta Bancaria** que permita crear cuentas con un número de cuenta y saldo inicial. Debe proporcionar métodos para depositar, retirar y consultar saldo.

Requisitos:

- Los atributos deben ser privados.
- Los métodos deben controlar que el saldo nunca sea negativo.

Preguntas:

- ¿Qué atributos debería tener la clase? Numero de cuenta y Saldo
- ¿Cómo se asegura que el saldo no se vuelva negativo? El saldo de una cuenta no puede ser negativo para evitar errores lógicos, por lo tanto, validamos que el saldo de la cuenta sea mayor a cero.

2. Control de Temperatura

Descripción: Crea una clase Termómetro que almacene la temperatura en grados Celsius y permita convertirla a Fahrenheit y Kelvin.

Requisitos:

- Los atributos deben ser privados.
- Se deben crear métodos para obtener y establecer la temperatura.

Preguntas:

- ¿Cómo se aplica la encapsulación en este problema?

Se aplica al declarar el atributo temperaturaCelsius como privado. Esto asegura que la temperatura solo pueda ser accedida y modificada a través de los métodos getTemperaturaCelsius y setTemperaturaCelsius. De esta manera controlamos el acceso y la modificación de la clase.

¿Cómo se realiza la conversión de temperatura en los métodos? La conversión de temperatura se realiza en los métodos convertirAFahrenheit y convertirAKelvin:

Fahrenheit: La fórmula para convertir de Celsius a Fahrenheit es (temperaturaCelsius×95)+32(temperaturaCelsius×59)+32.

Kelvin: La fórmula para convertir de Celsius a Kelvin es temperaturaCelsius+273.15temperaturaCelsius+273.15.

3. Registro de Productos en un Inventario

Descripción: Diseña una clase Producto que represente un producto en una tienda. Debe permitir establecer un código único, nombre y precio.

Requisitos:

- Usar **métodos getter y **setter para acceder y modificar el precio.
- No debe permitirse precios negativos.

Preguntas:

- ¿Cómo se encapsula la información del producto? Esto pasa cuándo agregamos los atributos código, nombre y precio como privados. Esto asegura que estos atributos solo puedan ser accedidos y modificados a través de los métodos públicos getPrecio, setPrecio, getCodigo y getNombre.
- ¿Por qué es importante validar los valores ingresados? Es importante validar los valores ingresados para asegurar que los datos del producto sean consistentes y válidos. En este caso, no permitir precios negativos evita errores lógicos y asegura que los productos tengan precios realistas.

4. Temporizador con Alarma

Descripción: Implementa un sistema de temporizador que permita contar el tiempo en segundos y activar una alarma cuando llegue a un tiempo específico.

Requisitos:

- Debe haber una clase Temporizador con atributos privados para el tiempo actual y un método para iniciarlo.
- Se debe crear una clase Alarma, que almacene un tiempo objetivo y se active cuando el temporizador lo alcance.
- El Temporizador debe permitir asociar una Alarma y verificar si debe activarse.

Preguntas:

- ¿Cómo interactúan las clases Temporizador y Alarma? Temporizador tiene un atributo alarma que puede ser asociado mediante el método asociarAlarma.

Durante la ejecución del método iniciar, Temporizador verifica si la alarma debe activarse llamando al método debeActivarse de la instancia Alarma.

- ¿Cómo se asegura que los atributos sean accesibles solo mediante métodos específicos**?** La encapsulación se asegura al declarar los atributos tiempoActual y alarma en Temporizador, y tiempoObjetivo en Alarma como privados. Esto garantiza que estos atributos solo puedan ser accedidos y modificados a través de métodos específicos, controlando así el acceso y modificación del estado interno de los objetos.

5. Control de Notas de un Curso

Descripción: Diseña un sistema que permita registrar estudiantes y calcular el promedio de un curso.

Requisitos:

- La clase Estudiante debe contener atributos privados para nombre, carnet y nota final.
- La clase Curso debe contener una lista de Estudiante y un método para calcular el promedio del curso.
- No se deben permitir notas fuera del rango 0 - 100.
- Se debe proporcionar un método para determinar qué estudiantes aprobaron (nota ≥ 61).

Preguntas:

- ¿Cómo se encapsulan los datos de los estudiantes dentro del curso? Declaramos los atributos nombre, carnet y nota como privados, para poder acceder desde los métodos getter y setter para que estos puedan obtener y asignar valores específicos.
- ¿Cómo se garantiza que las notas sean válidas? Las notas validas deben de ser mayor a cero y menor a cien, esto para que no exista errores lógicos.

Entregables:

- Colocar los archivos .java (**solamente los archivos .java**) en un repositorio público en Github.
- Colocar el enlace al repositorio como entregable de esta actividad.

Fecha de entrega: viernes 28 de febrero antes de las 23:59 hrs.