# 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 **CuentaBancaria** 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 para almacenar la información de la cuenta.
   - ¿Cómo se asegura que el saldo no se vuelva negativo?

Se asegura el constructor usando Math.max(saldoInicial, 0), y en el método retirar, verificando que el saldo sea suficiente antes de permitir el retiro.

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 encapsula la temperatura al hacer que el atributo temperaturaCelsius sea privado y proporcionar métodos setTemperatura() y getTemperaturaCelsius() para acceder al valor.
   - ¿Cómo se realiza la conversión de temperatura en los métodos?
     Fahrenheit: (temperaturaCelsius * 9/5) + 32

Kelvin: temperaturaCelsius + 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?
     Se encapsulan los atributos codigo, nombre y precio al hacerlos privados y proporcionando métodos getPrecio() y setPrecio().
   - ¿Por qué es importante validar los valores ingresados?
     Para evitar datos innecesarios, como precios negativos, lo cual podría causar errores en cálculos

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?
     La clase Temporizador tiene un objeto Alarma asociado. Mientras el temporizador avanza, verifica si el tiempo actual coincide con el tiempo objetivo de la alarma.
   - ¿Cómo se asegura que los atributos sean accesibles solo mediante métodos específicos?
     Los atributos tiempoActual y tiempoObjetivo son privados, y solo se pueden modificar o consultar a través de métodos públicos como verificarAlarma() y iniciar().

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?
  Se encapsulan dentro de la clase Estudiante, donde los atributos nombre, carnet y notaFinal son privados. La clase Curso solo accede a las notas a través de métodos públicos.
- ¿Cómo se garantiza que las notas sean válidas?
  Se valida en setNotaFinal(), asegurando que la nota esté en el rango 0-100. Si el valor no es válido, no se actualiza la nota.

**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.**