Seguimiento 1

Ejercicio 1:

Contexto

Una empresa alquila **lavadoras por hora**. El sistema debe calcular el **costo total** para un cliente según las horas alquiladas. Si el cliente alquila **más de 5 horas**, obtiene un **descuento del 3%** sobre el subtotal.

Objetivo

Construir una solución en Java que:

1. Lea (o reciba) la **tarifa por hora** y la **cantidad de horas**.
2. Calcule **subtotal**, **descuento** (si aplica) y **total a pagar**.
3. Muestre los resultados al usuario.

Reglas de negocio

- **Tarifa por hora**: valor positivo (por ejemplo, en pesos).
- **Horas alquiladas**: entero o decimal positivo (según decidan; mínimo > 0).
- **Descuento**:
  - Si **horas > 5 ⇒ 3%** del subtotal.
  - En otro caso ⇒ **0%**.
- **Subtotal** = tarifaPorHora × horas.
- **Total** = subtotal − descuento.

Requisitos de implementación (estructura)

- Crear una **clase base** (por ejemplo, `AlquilerLavadora`) con al menos:
  - **Atributos**: `tarifaPorHora`, `horas`.
  - **Métodos** mínimos:
    - `calcularSubtotal()`
    - `calcularDescuento()` (usa **if** o **operador ternario**)
    - `calcularTotal()`
- Crear una clase con `public static void main(String[] args)` (por ejemplo, `App`):
  - Instanciar `AlquilerLavadora`.
  - Asignar valores de prueba a `tarifaPorHora` y `horas` (o leerlos por consola).
  - Invocar los métodos de cálculo.
  - Imprimir **subtotal**, **descuento** y **total**.

Ejercicio 2:

Contexto

Se desea simular el proceso de **recarga de saldo para un teléfono celular**.
El sistema debe permitir registrar:

- **Número de celular**
- **Operador de telefonía**
- **Valor a recargar**
- **Saldo actual**

Además, el sistema debe ofrecer **dos opciones**:

1. Consultar el saldo actual.
2. Realizar una recarga.

Objetivo

Construir una solución en Java que:

1. Solicite o reciba el **número de celular**, **operador** y **saldo inicial**.
2. Permita al usuario elegir entre:
    - **Consultar saldo** (mostrar el saldo actual).
    - **Recargar** (aumentar el saldo según el valor ingresado).
3. Si se realiza la recarga, mostrar un mensaje de confirmación con todos los datos de la operación.

Reglas de negocio

- El número de celular debe tener un formato válido (puede simplificarse como una cadena de 10 dígitos para este ejercicio).
- El valor a recargar debe ser positivo.
- La opción seleccionada debe ser **1** o **2**.
- No se permite recargar valores negativos o cero.

Requisitos de implementación (estructura)

- Crear una **clase base** (por ejemplo, `RecargaCelular`) con:
    - **Atributos**: `numeroCelular`, `operador`, `saldo`.
    - **Métodos**:
        - `consultarSaldo()` → imprime el saldo actual.

        - `recargar(double valor)` → suma al saldo y muestra mensaje de recarga exitosa (número, operador, valor recargado, saldo final).
- Crear una clase con `public static void main(String[] args)` (por ejemplo, `App`):
    - Crear instancia de `RecargaCelular` con datos de ejemplo o leídos por consola.
    - Permitir elegir opción (1 o 2).
    - Ejecutar el método correspondiente.

Ejercicio 3:

Contexto

La Alcaldía de **Armenia** otorga **beneficios económicos** a ciudadanos registrados en el **Sisbén**.
Primero se verifica si la persona **pertenece al Sisbén**. Si **sí**, se asigna un beneficio base según **género** y, además, si la persona tiene **más de 60 años**, recibe un **adicional**.

Reglas de negocio

- El beneficio **solo se otorga** a quienes **pertenecen al Sisbén**. Si **no pertenece**, el total es **$0**.
- Si pertenece al Sisbén:
    - **Mujer ⇒ $150.000** de beneficio base.
    - **Hombre ⇒ $120.000** de beneficio base.
    - **Adicional por edad**: si **edad > 60 ⇒ +$40.000** (sin distinción de género).
- Total a entregar = beneficio base (según género) + adicional por edad (si aplica).

Objetivo

Construir una solución en Java que:

1. Reciba/lea **perteneceSibsen** (boolean), **género** (Mujer/Hombre) y **edad** (entero).
2. Calcule **beneficio base**, **adicional** y **total** usando **if** u **operador ternario**.
3. Muestre claramente los valores calculados al usuario.

Requisitos de implementación (estructura)

- Crear una **clase base** (ej., `BeneficioAlcaldia`) con:
    - **Atributos**: `perteneceSisben` (boolean), `genero` (String: "Mujer"/"Hombre"), `edad` (int).
    - **Métodos** mínimos:
        - `calcularBeneficioBase()`
        - `calcularAdicionalEdad()`
        - `calcularTotal()`
          *(Usar **if** o **ternario** en al menos uno de estos métodos).*
- Crear una clase con `public static void main(String[] args)` (ej., `App`):
    - Instanciar `BeneficioAlcaldia` con datos de ejemplo (o leer por consola).
    - Invocar los métodos de cálculo.
    - Imprimir **base**, **adicional** y **total**.

Ejercicio 4:

Contexto

Una empresa que vende **combos de hamburguesa** otorga **puntos de fidelización** según el **monto de compra**:

- Si la compra es **> 100.000** y **< 500.000** pesos ⇒ **100 puntos**.
- Si la compra es **> 500.000** pesos ⇒ **400 puntos**.
- En cualquier otro caso ⇒ **0 puntos**.
  La persona puede **consultar sus puntos** acumulados.

Objetivo

Construir una solución en Java que:

1. Reciba el **monto de una compra**.
2. **Asigne puntos** según las reglas y los **acumule** al cliente.
3. Permita **consultar los puntos** actuales del cliente.

Reglas de negocio

- Límite inferior y superior son **estrictos** (no incluyen 100.000 ni 500.000).
- Compras con valor **≤ 0** no generan puntos (y deben rechazarse o contar 0).
- Los puntos se **acumulan** por cada compra válida.

Requisitos de implementación (estructura)

- Crear una **clase base** (p. ej., `ClienteFidelizacion`) con:
  - **Atributos**: `nombre` (String), `puntos` (int).
  - **Métodos**:
    - `registrarCompra(double monto)` → calcula los puntos a otorgar con **if** o **ternario**, los suma a `puntos` y devuelve los puntos otorgados.
    - `consultarPuntos()` → retorna/imprime los puntos actuales.
- Crear una clase con `public static void main(String[] args)` (p. ej., `App`):
  - Instanciar `ClienteFidelizacion`.
  - Registrar **una o varias compras** (pueden ser valores de prueba o leídos por consola).
  - Mostrar los **puntos otorgados** por compra y el **total acumulado** al final.

Ejercicio 5:

Contexto

Un colegio organiza el **pago de matrícula** usando “pico y placa” según el **último dígito de la cédula**:

- Cédulas terminadas en **0 a 5**: pagan **lunes y miércoles**.
- Cédulas terminadas en **6 a 9**: pagan **martes y jueves**.
- Además:
    - Si la cédula es **par** ⇒ también **viernes**.
    - Si la cédula es **impar** ⇒ también **sábado**.

Objetivo

Construir una solución en Java que, dada una **cédula**, determine y muestre **los días** en que la persona puede pagar la matrícula.

Reglas de negocio

- La cédula debe ser numérica (cadena de dígitos).
- **Último dígito**:
    - **0–5** ⇒ lunes, miércoles.
    - **6–9** ⇒ martes, jueves.
- **Paridad** de la cédula completa:
    - **Par** ⇒ se agrega **viernes**.
    - **Impar** ⇒ se agrega **sábado**.
- La salida debe listar **todos los días** aplicables (base + adicional por paridad).

Requisitos de implementación (estructura)

- Crear una **clase base** (ej.: `PicoPlacaMatricula`) con:
    - **Atributos**: `cedula` (String).
    - **Métodos** mínimos:
        - `obtenerUltimoDigito()` → retorna `int`.
        - `esPar()` → retorna `boolean` (según la cédula completa).
        - `diasBase()` → retorna un texto con los días base **usando if o ternario**.
        - `diaAdicional()` → retorna “viernes” o “sábado” según paridad.
        - `diasPermitidos()` → construye el texto final con **todos** los días.
- Crear una clase con `public static void main(String[] args)` (ej.: `App`):
    - Instanciar `PicoPlacaMatricula` con una cédula (leer por consola o asignar).
    - Invocar `diasPermitidos()` y mostrar el resultado.