Simulación de un Gran Premio de Carreras de Caballos

1) Contexto

Vas a implementar una mini-aplicación en Java que simula un **Gran Premio** compuesto por varias **carreras**. En cada carrera compiten **caballos** montados cada uno de ellos por un **jinete**. Habrá **apostantes** que realizan apuestas antes de cada carrera, siempre que tengan saldo suficiente. Al finalizar todas las carreras del Gran Premio se mostrará el saldo final de cada apostante.

2) Objetivo general

Diseñar y programar un conjunto de clases e interfaces que:

- Simulen el avance por **turnos** de los caballos (con componente aleatoria).
- Gestionen **apuestas** y **saldos** de apostantes.
- Determinen **ganadores**, **actualicen experiencia** y **paguen premios**.
- Usen **herencia**, **métodos/ clases abstractas**, **interfaces**, **listas**, **bucles**, **condicionales**, **Strings**, **métodos**, y **métodos estáticos**.

3) Reglas de la simulación

1. Un **Gran Premio** tiene un nombre y **N carreras** (N ≥ 2).
2. Cada **Carrera** se define por un nombre una **distancia objetivo** (en metros) y una **lista de caballos participantes**.
3. Antes de que empiece cada carrera:
    - Cada **Apostante** puede realizar **1** apuesta, siempre que tenga **saldo suficiente** y nunca podrá apostar más del saldo que tiene.
    - Una apuesta consiste en: *apostante*, *caballo elegido*, *importe*.
4. Desarrollo de la carrera (por **turnos**):
    - En cada turno, **cada caballo** avanza una cantidad de metros calculada como:
    - avanceTurno = baseAleatoria + (velocidad) + (experiencia) - (peso) + aniosExperienciaJinete, donde baseAleatoria [0.0, 10.0) proviene de Math.random().
        - **peso** resta (penaliza).

        - **velocidad** y **experiencia** y **aniosExperienciaJinete** suman (bonifican).
        - El avance mínimo por turno no puede ser negativo (si la fórmula diera negativo, usar 0).
    - El primer caballo que **alcance o supere** la distancia de la carrera que esté corriendo es el **ganador** de la carrera.
5. Consecuencias:
    - El **caballo ganador** aumenta su **experiencia** (p. ej., +1 punto).
    - Las **apuestas** al caballo ganador pagan **importe × 5** al apostante.
    - Las apuestas perdedoras se pierden.
6. Al finalizar **todas** las carreras del **Gran Premio**, se muestra:
    - **Clasificación** de cada carrera (al menos ganador).
    - **Saldo final de cada apostante**.

4) Modelo orientado a objetos (obligatorio)

4.1 Clases principales

- **Caballo**
    - Atributos mínimos: String nombre, double peso, double velocidad, int experiencia, Jinete jinete, double metrosRecorridos (reseteable por carrera).
    - Métodos clave: getters/setters, void resetear(), void sumarExperiencia(int puntos).
- **Persona**
    - Atributos mínimos: String nombre, int edad.
- **Jinete**
    - Atributos mínimos: int aniosExperiencia.
- **Apostante**
    - Atributos mínimos: double saldo.
- **Carrera**
    - Atributos mínimos: String nombre, double distanciaObjetivo, List<Caballo> participantes, List<Apuesta> apuestas.

- **GranPremio**
    - Atributos mínimos: String nombre, List<Carrera> carreras, List<Apostante> apostantes.
    - Métodos: void empezarGranPremio(), void mostrarResumen().
- **Apuesta**
    - Atributos: Apostante apostante, Caballo caballo, double importe.

4.2 Herencia y clases/ métodos abstractos

- Clase Persona que será la clase padre de Jinete y Apostante
- Define una **clase abstracta** Participante que herede de Persona con, el método String getIdentificador() (abstracto).
    - **Caballo** y **Jinete** deben **heredar** de Participante.
    - Implementa getIdentificador() adecuadamente en cada subclase.

4.3 Interfaces

- Crea una interfaz Avanzable con:
    - double calcularAvanceTurno()
    - void aplicarAvance(double metros)
- Haz que **Caballo** implemente Avanzable.
- Crea una interfaz Imprimible con:
    - String imprimeDatos() (pinta los datos del objeto que lo tenga).
- Implementa Imprimible al menos en **Caballo** y **Apostante**.

4.4 Colecciones

- Todas las relaciones “muchos” se gestionan con List<> (por ejemplo ArrayList<>).
- Deberás recorrer colecciones con **bucles** y tomar decisiones con **condicionales** (if/else, switch opcional).

5) Cálculo del avance por turno (detalle)

- Implementa el cálculo dentro de calcularAvanceTurno():

- Usa Math.random() para la parte aleatoria [0.0, 10.0).
- Asegura avanceTurno = baseAleatoria + velocidad + experiencia - peso) o 0 si es negativo.
- El método **no** modifica metrosRecorridos; esa responsabilidad será de aplicarAvance.

- En Carrera tenemos el método iniciar():
  - Inicializa metrosRecorridos = 0 para todos los caballos.
  - Repite **turnos** hasta que algún caballo alcance distanciaObjetivo.
  - (Opcional) Controla empates: si dos o más superan la distancia en el mismo turno, el ganador es el que **más metros** haya acumulado. Si persiste el empate, el ganador será el de **mayor experiencia**; si aún empatan, el de **menor peso**; si aún empatan, el que **aparezca antes** en la lista.
  - **Crear y llamar al método Registra** (String) eventos relevantes por turno (opcional pero recomendado para depuración: "Turno 3: Furia: +2.41m (total 98.7m)").

6) Gestión de apuestas

- Al registrar una apuesta:
  - Validar que importe > 0 y que el **apostante** puedeApostar.
  - Al **confirmar** la apuesta, **descuenta** el importe del saldo del apostante.
- Tras determinar el **ganador**:
  - Recorre las apuestas:
    - Si caballo == ganador, abona importe × 5 al apostante.

7) Clase de utilidades con métodos estáticos (obligatorio)

Crea una clase SimUtils con **solo métodos estáticos**:

- static Caballo crearCaballoAleatorio(String nombreBase, Jinete j) → devuelve un caballo con atributos aleatorios razonables (peso, velocidad, experiencia inicial).
- Métodos necesarios para recoger los datos introducidos por consola

8) Entrada/Salida y requisitos de ejecución

- **Sin entrada por teclado obligatoria**: puedes **crear** objetos en main (crea 3–5 caballos, 2–3 carreras y 2–3 apostantes).
- Muestra por consola:
    1. **Inicio del Gran Premio XXX**
    2. Para cada carrera mostrar: distancia, participantes y sus atributos.
    3. Apuestas registradas (nombre apostante, caballo, importe).
    4. Desarrollo resumido (turnos opcional) y **ganador**.
    5. **Saldo final** de cada apostante al concluir el Gran Premio (formateado).

11) Criterios de evaluación (propuestos)

1. **Diseño OO (30%)**: uso correcto de herencia, abstracciones e interfaces; cohesión y bajo acoplamiento.
2. **Funcionalidad (35%)**: simulación por turnos correcta, cálculo de avance, resolución de empates, actualización de experiencia, pago de apuestas, saldos finales.
3. **Uso de colecciones, bucles y condicionales (15%)**: recorridos, filtros y decisiones.
4. **Calidad de código (10%)**: constantes, validaciones, toString(), legibilidad, logs.
5. **Métodos estáticos útiles (10%)**: diseño y uso efectivo de SimUtils.

12) Entregables

- Proyecto Java (Maven) con **paquetes** organizados subido a GitHub.
- URL GitHub:

- avanzan por turno
- avanzan todos juntos
    - mostrar lo que han avanzado
    - mostrar su recorrido (hasta el momento)
- mostrar solo el ganador o el ranking de los caballos