Práctica:
Videojuego de Batalla por Turnos en Java *(NetBeans, Ant, Consola)*

Objetivo General

Desarrollar un **videojuego de combate por turnos en consola**, aplicando conceptos de **herencia**, **estructuras de datos**, **controles de flujo** y **persistencia en base de datos**. El estudiante debe demostrar dominio en la programación orientada a objetos, el manejo de menús, la lógica de turnos y el uso de una base de datos relacional para almacenar información relevante del juego.

Descripción General

Desarrollar en **Java (NetBeans, usando Ant, modo consola)** un videojuego para dos jugadores *(trabajo en parejas)*, donde cada uno podrá seleccionar un personaje de una raza específica: **Humano, Elfo, Orco, Bestia**.

El combate se desarrolla por turnos. Cada raza tiene armas y habilidades únicas. Todos los personajes descienden de una **clase padre llamada *Personaje***.

Se requiere persistir la información de las clases principales en una base de datos: personajes, razas, armas y jugadores (incluyendo el total de partidas ganadas y perdidas).

Requisitos Específicos

1. Selección de Personaje y Menú de Juego

- Al iniciar, cada jugador debe ingresar su **nombre** y seleccionar una raza usando un **menú por consola** (switch).
- *Razas disponibles:*
    - **Humano**: sólo puede usar armas de fuego (escopeta o rifle francotirador).
    - **Elfo**: sólo puede usar magia (báculo). Puede elegir entre magia de fuego, tierra, aire o agua.
    - **Orco**: sólo puede usar hacha o martillo.

- **Bestia (híbrido animal con otra raza)**: puede elegir entre puños o espada.

2. Lógica de Combate y Armas

- El combate es **por turnos** (primero el Jugador 1, luego el Jugador 2).
- **Cada jugador inicia con 100 puntos de vida**. *Excepción*: el elfo con magia de agua inicia con 115 puntos.
- Cada arma o hechizo afecta la vida del oponente según las siguientes reglas:
  - **Humanos**
    - *Escopeta*: daño aleatorio 1-5 +2%.
    - *Rifle francotirador*: daño aleatorio 1-5.
    - *Sanación*: pueden gastar un turno en **comer**, recuperan 50% del daño recibido.
  - **Elfos**
    - *Fuego*: daño aleatorio 1-5 +10%.
    - *Tierra*: daño aleatorio 1-5 +2%, acierta más veces.
    - *Aire*: daño aleatorio 1-5.
    - *Agua*: daño aleatorio 1-5, la sanación aumenta a 90%.
    - *Sanación*: pueden gastar turno, curan 75% (agua: 90%) con **hechizo de sanación**.
    - *Solo el elfo con magia de agua inicia con 115 puntos de vida*.
  - **Orcos**
    - *Hacha*: daño aleatorio 1-5, además provoca 2 turnos de *sangrado* (-3 vida*turno).
    - *Martillo*: daño aleatorio 1-5.
    - *Sanación*: gastan turno, curan 25% **pócima de curación** y 15% adicional el siguiente turno.
  - **Bestias**
    - *Puños*: daño fijo 25 al oponente, pero pierde 10 de vida el atacante.
    - *Espada*: daño aleatorio 1-10.
    - *Sanación*: gastan turno en **dormir**, curan 45% de vida.

3. Persistencia en Base de Datos

- **Bases de datos permitidas**: **PostGreSQL**, **MS SQL Server** u **Oracle** (23ai o XE 21c).

- Usar JDBC (según el gestor seleccionado).
- **Tablas requeridas**:
    - personaje: id, nombre, raza, fuerza, energía, vida_actual, id_arma
    - raza: id, nombre, descripción
    - arma: id, nombre, tipo, daño_mínimo, daño_máximo, modificadores
    - jugador: id, nombre, partidas_ganadas, partidas_perdidas
- Al finalizar una partida, registrar en la base de datos:
    - Los personajes utilizados, armas, raza, resultado (quién ganó, quién perdió), y actualizar estadísticas del jugador.

Opcional (Puntos Extra)

- Implementar la **lógica de avance y retroceso**: para atacar, ambos jugadores deben estar frente a frente, perdiendo turnos para avanzar o retroceder.
- Si se encuentran a distancia:
    - *Rifle francotirador*: su daño sube de 5 a 10 (aleatorio).
    - *Magia de aire*: daño sube de 4 a 12 (aleatorio).
- **Ambiente gráfico**: quienes implementen el juego en ambiente gráfico (Swing o JavaFX), recibirán **puntos extra**.

Requisitos Técnicos

- Todo el código debe estar organizado en **paquetes**.
- El proyecto debe compilar y ejecutarse desde NetBeans (con Ant).
- Se deben utilizar **herencia** (clase padre Personaje, clases hijas para cada raza).
- Se deben aplicar correctamente **estructuras de datos** (listas, arrays o colecciones).
- Se debe utilizar **switch** y otros **controles de flujo**.
- Deben implementarse métodos de ataque, defensa y sanación.
- Los mensajes y menús deben estar claros y guiados para el usuario.

Entregables

1. **Proyecto Java completo** en una carpeta comprimida (.zip), con toda la estructura y base de datos incluida.

2. **Script SQL** de la base de datos y el archivo de respaldo **.bak** (SQL Server u Oracle) o **.backup** (PostgreSQL) de la base de datos llena, con ejemplos.
3. **Repositorio en GitHub** con el proyecto y al menos **4 commits** por cada miembro.
4. **README.md** como manual de usuario, donde expliquen:
    - Cómo jugar.
    - Cómo ejecutar el juego.
    - Detalles técnicos y requisitos.
    - Breve descripción del sistema y roles de cada integrante.
    - *Incluyan capturas de pantalla si es posible*.

Criterios de Evaluación

Criterio	Puntaje
Herencia aplicada correctamente	20
Estructuras de datos y flujo de control	15
Funcionalidad completa (todas las razas y armas, lógica de combate, sanaciones, persistencia)	30
Persistencia en base de datos (tablas, operaciones CRUD, respaldo)	15
Menús claros, código comentado y organizado	10
Manual de usuario en README.md y repositorio GitHub	10
Puntos Extra: Ambiente gráfico u opciones avanzadas	**+10**

Total: 100 puntos + Extra

Notas Finales

- El trabajo es **únicamente en parejas**. Ambos deben participar y conocer el funcionamiento del código.
- Se permite y recomienda la creatividad en el diseño de menús y la historia de los personajes.
- Se valorará el uso de buenas prácticas de programación y la limpieza del código.
- **Plagio parcial o total** implica nota 0.
- **La próxima clase (sábado)**, deberán **presentar el juego funcionando**, defender su código y explicar cómo implementaron la solución.