**Explicación clases de personajes y enemigos:**

Dentro de nuestro sistemas contamos con dos tipos de personajes: jugador y enemigo

Como los objetos Jugador como los objeto Enemigo comparten varios atributos se ha creado una clase Personaje, que se pasa a explicar a continuación:

Personaje:

* + - Atributos:
      * + private String nombre
        + private int nivel
        + private int hp
        + private int hpActual
        + private int ataque —> En el caso de jugadores será la suma del atributo “ataqueBase” (que tiene la clase Jugador) más el ataque del arma
        + private int defensa —> Igual que anterior pero con atributo “defensaBase” y la defensa que proporciona la armadura
        + private int velocidad —> Utilizado para generar a la hora de combatir el orden de ataque tanto de enemigos como de jugadores
        + private Imagen imagen
    - Métodos:
      * + Getters and Setters para la utilización de dichos atributos
        + public boolean estaVivo() —> Usado para saber si un personaje esta vivo

Ahora se pasa a explicar la clase Jugador, la cual es padre a su vez de las clases Horacia, Kibito, Mordeim y Archienemigo que son los tipos de personajes que contiene el juego. También es una clase abstracta porque las diferentes clases hijas tendrán unos métodos abstractos que cada uno implementará de una manera

Jugador:

* + - Atributos:
      * + private int mp
        + private int mpActual
        + private int habCritico
        + private Arma arma
        + private Armadura armadura
        + private ArrayList<Habilidad> habilidades —> Cada jugador tiene 5 habilidades, distribuidas a lo largo de sus niveles
        + private int exp
        + private int expProxNivel
        + private Inventario inventario
        + private int defensaBase
        + private int ataqueBase
    - Métodos:
      * + Getters and Setters usados para manejo de atributos anteriores
        + public boolean cambiarArma(Arma arma)
        + public boolean cambiarArmadura(Armadura armadura)
        + public void anadirHabilidad(Habilidad hab)
        + public boolean puedeSubir() —> Comprueba si el jugador puede subir
        + public void subirNivel()
        + public boolean habilidadUsable(int nHabilidad)
        + private boolean comprobarLVL(int nHabilidad)
        + private boolean comprobarMp(int nHabilidad)
        + public boolean usarHabilidad(int nHabilidad, Personaje objetivo) —> Método  
          sobrecargado para usar habilidad contra un enemigo
        + public boolean usarHabilidad(int nHabilidad, ArrayList<Personaje> objetivos) —>   
          Método sobrecargado para usar una habilidad contra varios enemigos
        + public abstract void inicializarPersonaje() —> Como cada jugador se iniciará de una   
          manera se tiene este método abstracto
        + public abstract void subirNivelEstadisticas() —> Es abstracto porque cada jugador sube las estadísticas de manera distinta
        + public abstract void setearHabilidades()

Las clases hijas de Jugador lo que harán serán implementar los métodos abstractos de la manera que se especifique

Por último, la clase Enemigos es a su vez padre de todos los tipos de enemigos como por

ejemplo MiniGrifo, Goblin, Rata, etc. También es abstracta porque tienen varios métodos abstractos

Enemigo:

* + - Atributos:
      * + private int exportada
        + private int oro
        + private int id
        + private ArrayList<Habilidad> habilidades
    - Métodos:
      * + Getters and Setters usados para manejo de atributos anteriores
        + public String escribirMensaje(boolean habilidad, Habilidad hab, Jugador jugador, int danyo) —> Usado para mostrar en la interfaz de Combate un mensaje sobre el ataque de enemigo
        + public abstract String estrategiaAtacar(ArrayList<Jugador> jugadores) —> Implimentará la estrategia que tiene el tipo de enemigo contra los aliados, es decir, atacar aleatorio, atacar al primero con vida, etc.
        + public abstract void inicializarEnemigo() —> Sirve para inicializar al tipo de enemigo

Las clases hijas de Enemigo lo que harán serán implementar los métodos abstractos de la

manera que se estime oportuno