25. Polimorfismo

1. **Ejercicio 1: La Orquesta Sinfónica (Nivel Básico)** 🎻

   Vamos a crear un programa que simule una orquesta. Cada instrumento musical suena de una forma distinta, pero todos responden a la orden de "tocar".

   La superclase: Crea una clase padre llamada Instrumento. Dentro, define un método llamado tocar() que de momento solo tenga la instrucción pass.

   Las subclases: Crea tres clases hijas que hereden de Instrumento: Guitarra, Tambor y Piano.

   En cada clase hija, sobrescribe el método tocar() para que imprima un sonido diferente (por ejemplo: "Plin plin plin" para el piano, "Pom pom pom" para el tambor...).

   En el programa principal, crea un objeto de cada instrumento. Mételos todos juntos en una lista llamada orquesta.

   Recorre la lista orquesta con un bucle for y dale a cada instrumento la orden de tocar().

2. **Ejercicio 2: El Ataque del Equipo (Nivel Intermedio)** ⚔️

   Vamos a programar el sistema de combate de nuestro propio videojuego de Rol (Pythonator). Tenemos diferentes tipos de personajes, y todos saben atacar, pero cada uno usa su propio estilo.

   Clase Padre: Crea la clase Personaje. Su método constructor (__init__) debe recibir y guardar el nombre del personaje. Además, crea un método atacar() vacío (con pass).

   Clases Hijas: Crea las clases Guerrero, Mago y Arquero (todas heredan de Personaje).

   Ataques personalizados: Sobrescribe el método atacar() en cada una.

   - El Guerrero debe imprimir: "¡[Nombre del guerrero] ataca con un espadazo!"
   - El Mago debe imprimir: "¡[Nombre del mago] lanza una bola de fuego!"
   - El Arquero debe imprimir: "¡[Nombre del arquero] dispara una flecha certera!"

   Crea tres personajes (por ejemplo: Arturo el guerrero, Merlín el mago y Legolas el arquero).

   Introduce todos los objetos en una lista llamada equipo y usa un bucle for para ordenarles a todos que ejecuten su método atacar().

3. **Ejercicio 3: El Contrato Estricto (Nivel Profesional)** 🏢

   En este ejercicio vamos a aprender cómo obligar a que nuestro código sea seguro.

   Imagina que estamos diseñando el sistema de empleados de una empresa. Creamos la clase padre Empleado con un método trabajar().

   Hasta ahora usábamos pass en la clase padre, pero si a un programador de nuestro equipo se le olvida escribir el método trabajar() en alguna de las clase hija, el programa fallará en silencio. Vamos a obligar a que cumplan las normas.

Crea la clase Empleado y en su método trabajar() lanza el siguiente error:

```
raise NotImplementedError("¡Error! Debes definir cómo trabaja este
empleado.")
```

Crea una clase hija Programador(Empleado). Define su método trabajar() para que imprima "Escribiendo código en Python...".

El error intencionado: Crea una clase hija Gerente(Empleado), pero NO le pongas el método trabajar() (finge que se te ha olvidado).

En el programa principal, crea un objeto Programador y ordénale trabajar(). Comprueba que funciona.

Crea un objeto Gerente y ordénale trabajar(). ¡Observa cómo Python detiene el programa y te lanza el error de seguridad que tú mismo diseñaste al crear la superclase.