# CASO PRÁCTICO 1

**DESARROLLO DE CLASES**

## Contexto

El jefe de proyecto de nuestro equipo de programación nos pide realizar un sistema que simule el mundo de los Pokémon, empleando la programación orientada a objetos como paradigma fundamental y aprovechando los conceptos de asociación y herencia. Este sistema debe permitir modelar entrenadores y sus Pokémon principales, organizando combates simples entre ellos. Como desarrollador software en Java debemos tener clara la forma en la que se deben declarar y relacionar clases, así como el proceso de instanciación de objetos de estas clases.

## Cuestiones a resolver

Diseña un programa en Java que modele un sistema básico de combates Pokémon. Cada entrenador solo puede tener un único Pokémon que utiliza en el combate. Sigue estas indicaciones:

* Crea una **superclase** **Pokémon** que contenga 3 atributos comunes (invéntatelos)
* Define subclases o clases **hijas** que hereden de la clase Pokémon y que añadan al menos dos atributos y un método específico para cada clase hija, un ejemplo de clase hija puede ser *PokemonFuego*. Añade mínimo dos subclases sin contar el ejemplo dado, es decir, debes tener por lo menos tres clases hijas definidas.
* Crea una clase **Entrenador**, que represente a un entrenador de Pokémon, con atributos como su nombre y experiencia, debes añadir dos atributos más.
* Crea una clase **Combate**, que permita enfrentar a dos Pokémon en un duelo, comparando sus niveles para determinar un ganador, añade los atributos necesarios.
* Establece una relación de **asociación** entre Entrenador y Pokémon, donde cada entrenador tiene un único Pokémon que puede ser de los tipos indicados (clases hijas) asignado como compañero, añade a la clase Entrenador que acabas de realizar lo que consideres necesario para representar la relación propuesta.

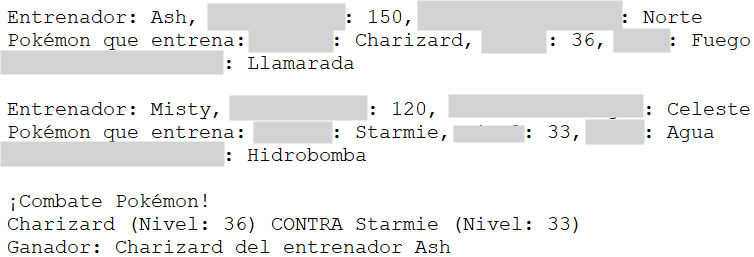
En el método main:

* Crea al menos dos entrenadores y asigna a cada uno su Pokémon.
* Organiza un combate entre los Pokémon principales de los entrenadores. El combate se basa en la comparación de los niveles de los Pokémon para decidir al ganador. Este es un enfoque simple, donde el Pokémon con el nivel más alto gana. Si los niveles son iguales, el combate termina en empate.
* Muestra por pantalla los datos de los entrenadores, los Pokémon participantes, y el resultado del combate (Pokémon ganador o empate).

Todas las clases deben contener todos los métodos getter y setter así como sus correspondientes constructores con parámetros y sin parámetros que deben ser utilizados de forma correcta.

Justifica la accesibilidad de los atributos y métodos de forma justificada en comentarios claros dentro del código

Ejemplo de pantalla de salida: (adáptala a las indicaciones del enunciado)

~~~~

Se debe entregar un solo archivo .java que contendrá todas las clases indicadas y un método main en el que se realizará la instanciación de objetos, inicialización y se mostrarán resultados al usuario.

## Recursos

Se deberá consultar el contenido de la unidad, internet, libros, revistas, etc.

En el apartado “Clasificación por tipos” de la siguiente web <https://vandal.elespanol.com/guias/guia-pokemon-go/que-son-los-pokemon> puedes encontrar ideas para tipos de Pokémon. Para la elaboración del programa se utilizará Eclipse como IDE de desarrollo.

## Objetivos

Dominar las estructuras de control del lenguaje de programación, como las estructuras de selección y repetición, para escribir y depurar código eficiente y funcional.

Desarrollar programas organizados en clases, aplicando los principios de la programación orientada a objetos para crear código modular, reutilizable y fácil de mantener.

Crear constructores para las clases, permitiendo la inicialización de objetos y facilitando su instancia y utilización en el código.

Implementar conjuntos y librerías de clases para organizar y reutilizar código en diferentes proyectos, mejorando la eficiencia y la calidad del desarrollo de software.

Aplicar la herencia en la creación de clases, aprovechando la reutilización de código y permitiendo establecer jerarquías entre las clases.

Aplicar la asociación en la creación de clases, facilitando la interacción entre objetos y modelando relaciones reales entre entidades de un sistema, como dependencias o colaboraciones.

## Resultados de aprendizaje y criterios de evaluación

Escribe y depura código, analizando y utilizando las estructuras de control del lenguaje.

* Se ha escrito y probado código que haga uso de estructuras de selección.
* Se han utilizado estructuras de repetición.
* Se han reconocido las posibilidades de las sentencias de salto.
* Se ha escrito código utilizando control de excepciones.
* Se han creado programas ejecutables utilizando diferentes estructuras de control.
* Se han probado y depurado los programas.
* Se ha comentado y documentado el código.
* Se han utilizado aserciones para la detección y corrección de errores durante la fase de desarrollo.

Desarrolla programas organizados en clases analizando y aplicando los principios de la programación orientada a objetos.

* Se ha reconocido la sintaxis, estructura y componentes típicos de una clase.
* Se han definido clases.
* Se han definido propiedades y métodos.
* Se han creado constructores.
* Se han desarrollado programas que instancien y utilicen objetos de las clases creadas anteriormente.
* Se han utilizado mecanismos para controlar la visibilidad de las clases y de sus miembros.
* Se han definido y utilizado clases heredadas.
* Se han creado y utilizado conjuntos y librerías de clases.

## Rúbrica para su evaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Excelente** | **Satisfactorio** | **Mejorable** | **Insuficiente** |
| **Creación de clases en Java.** | El alumno crea las clases indicadas con todos los atributos y métodos requeridos, incluyendo constructores con y sin parámetros, métodos getters y setters, y métodos funcionales específicos para cada clase, demostrando una comprensión avanzada de la programación orientada a objetos en Java. | Las clases están bien desarrolladas con la mayoría de los atributos y métodos necesarios, mostrando una buena comprensión de la estructura y funcionalidades de las clases en Java. | Las clases se crean con algunos atributos y métodos, pero pueden faltar detalles en los constructores, getters/setters o en los métodos funcionales. | Las clases están incompletas o incorrectamente implementadas, con omisiones significativas en atributos, métodos, y constructores, reflejando una comprensión deficiente de la programación orientada a objetos en Java. |
| 20% | 2 | 1.5 | 1 | 0.5 |
| **Especificación de accesibilidad de atributos y métodos.** | El estudiante especifica claramente y con justificación la accesibilidad (pública, privada, protegida) de cada atributo y método en las clases creadas, demostrando un entendimiento avanzado del encapsulamiento y la seguridad en Java. | Identifica correctamente la accesibilidad de la mayoría de los atributos y métodos con justificaciones adecuadas. | Ofrece una idea general de la accesibilidad de los atributos y métodos, aunque puede faltar en justificaciones o detalles. | La especificación de accesibilidad es inadecuada o incorrecta, con poca o ninguna justificación, indicando una comprensión limitada del control de acceso en Java. |
| 20% | 2 | 1.5 | 1 | 0.5 |
| **Relación entre clases.** | El alumno establece y explica con detalle y precisión las relaciones entre las clases creadas, utilizando código Java correcto y justificando el tipo de relación (herencia, composición, etc.). | Hay una buena explicación de las relaciones entre las clases, con código Java adecuado y justificaciones razonables. | Las relaciones entre clases están presentes, pero la explicación o el código Java pueden ser básicos o contener pequeños errores. | Las relaciones entre las clases son inadecuadas o mal explicadas, con código Java incorrecto o confuso. |
| 20% | 2 | 1.5 | 1 | 0.5 |
| **Instanciación y uso de clases.** | Crea instancias de cada clase usando ambos constructores de forma correcta, inicializa atributos adecuadamente y muestra los datos de cada objeto de manera eficiente y clara en la consola. | Realiza la instanciación y muestra los datos de cada objeto correctamente, aunque puede haber pequeñas ineficiencias o falta de claridad. | Las instancias de clase se crean y utilizan, pero pueden existir errores menores en la inicialización de atributos o en la visualización de datos. | La instanciación de clases o la visualización de datos es incorrecta o muy limitada, indicando una falta de comprensión en la creación y uso de objetos en Java. |
| 20% | 2 | 1.5 | 1 | 0.5 |
| **Código limpio con estructuras necesarias** | Presenta código limpio con estructuras necesarias y sin redundancia. | Presenta código limpio, pero hay algunas estructuras que se pueden simplificar, eliminar o tabular para evitar redundancia. | Presenta código más o menos limpio, pero hay estructuras que se pueden simplificar, eliminar o tabular para evitar redundancia. | Presenta código desordenado, hay muchas estructuras que se deben simplificar o eliminar para evitar redundancia. |
| **10%** | 1 | 0.75 | 0.5 | 0.25 |
| **Errores de compilación y/o ejecución.** | El programa se ejecuta sin errores de compilación o ejecución, realizando la función esperada. | El programa se ejecuta sin errores de compilación, pero sí con algunos errores en la ejecución, la función realizada varía ligeramente de la función esperada. | El programa se ejecuta sin errores de compilación, pero sí con varios errores en la ejecución, la función realizada varía de la función esperada. | El programa se ejecuta con errores de compilación por tanto no es ejecutable y/o el programa ejecuta sin errores de compilación, pero con numerosos errores en la ejecución, la función realizada no coincide con la función esperada. |
| **10%** | 1 | 0.75 | 0.5 | 0.25 |