Trabajo Práctico 2
Programación Orientada a Objetos

Paradigmas de Lenguajes de Programación — 2do cuat. 2021

Fecha de entrega: 28 de octubre de 2021

Introducción

Este trabajo práctico consiste en modelar Pokémon en JavaScript. Inicialmente representamos a cada Pokémon mediante un objeto que sabe responder a los mensajes `hp` y `ataque(Nombre del ataque)` para cada uno de los ataques que ese Pokémon conoce.

- Los Pokémon responden al mensaje `hp` mediante un valor entero que representa sus puntos de vida.
- Los Pokémon responden al mensaje `ataque(Nombre del ataque)` mediante una función que representa a ese ataque. Una función que representa un ataque toma como único parámetro a un objeto que representa un Pokémon (el oponente) al cual se va a atacar. Por ejemplo, un ataque podría decrementar el valor de *hp* del oponente.

Ejercicios

Ejercicio 1

Definir los objetos `pikachu` y `bulbasaur`. Estos deben poder responder al mensaje `hp`. El *hp* inicial de Pikachu es 250, y el de Bulbasaur es 300. Además, Pikachu puede atacar usando `ataqueImpactrueno` y Bulbasaur puede usar `ataquePlacaje` y `ataqueLatigoCepa`. Cada uno de estos tres ataques produce 10 *hp* de daño en el oponente.

Ejercicio 2

a) Definir el objeto `raichu`. Al ser su evolución, Raichu debe poder usar todos los ataques que Pikachu tenga definidos en la actualidad o que aprenda más adelante. Su *hp* inicial es 300, y tiene un ataque adicional, `ataqueGolpeTrueno`, que produce 30 *hp* de daño en el oponente.

b) Definir el objeto `pichu`. Su *hp* inicial es 100, y aunque actualmente no cuenta con ataques, en el futuro podría aprender algunos. Pikachu y Raichu deben poder usar los ataques que Pichu tenga definidos.

c) Vamos a agregar la información del tipo de los Pokémon. Escribir el código necesario para que los objetos que representan a cada uno de los Pokémon creados hasta ahora respondan al mensaje `tipo` con el string correspondiente. Pichu, Pikachu y Raichu son

de tipo eléctrico, mientras que Bulbasaur es de tipo planta. Algunos strings de tipo se encuentran definidos al principio de `taller.js`.

Ejercicio 3

a) Definir la función constructora `Pokemon` que permita crear un Pokémon a partir de un número que represente su *hp*, un objeto que represente sus ataques (donde la clave es el nombre del mensaje correspondiente, y el valor la función que representa ese ataque), y el string del tipo al que pertenece.

b) Definir el objeto `charmander` usando la función constructora. Su *hp* es 200, es de tipo fuego y ataca usando `ataqueAscuas`, que hace 40 *hp* de daño.

c) Agregar el código necesario para que los Pokémon definidos en los ejercicios anteriores se comporten como si hubieran sido creados mediante esta función constructora.

d) Agregar el código necesario para que todos los Pokémon puedan responder al mensaje `atacar` mediante un método que reciba como parámetros el string con el nombre de un ataque (por ejemplo, `''ataquePlacaje''`) y un objeto representando al Pokémon oponente. Si el Pokémon que recibió el mensaje puede usar el ataque pasado por parámetro, debe ejecutarlo sobre el oponente. En caso contrario, él mismo recibe 10 *hp* de daño.

Ejercicio 4

a) Agregar el código necesario para que todos los Pokémon puedan responder al mensaje `nuevoAtaque`, que recibe el nombre de un ataque y la función que lo representa, y lo agreguen a sus ataques disponibles.

b) Hacer que Pikachu aprenda el ataque `ataqueOndaTrueno`, que divide por 2 (redondeando hacia abajo) la *hp* del oponente. **Sugerencia**: ver la función `Math.floor()`: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/floor

Ejercicio 5

a) Agregar el código necesario para que todos los Pokémon puedan responder al mensaje `evolucionar`. Este debe devolver un nuevo objeto que representa al Pokémon resultante de su evolución, el cual tiene el mismo tipo que el original, el doble de *hp* y sabe realizar los mismos ataques.

b) Definir el objeto `charmeleon`, que es la evolución de Charmander, y `charizard`, que es la evolución de Charmeleon.

Ejercicio 6

a) Agregar el código necesario para que todos los Pokémon puedan responder al mensaje `algunAtaque`. Este método debe devolver aleatoriamente el nombre de alguno de los ataques que este Pokémon pueda realizar. Por ejemplo, `raichu.algunAtaque()` debería poder devolver `'ataqueImpactrueno'`. No está definido el comportamiento si el Pokémon no sabe ningún ataque. **Sugerencia**: ver la función `Math.random()`: https://

developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random

b) Definir la función `peleaPokemon`. Esta debe recibir dos Pokémon como parámetro; el primero es el que ataca primero. Asumimos que ambos tienen *hp* mayor a 0 y saben realizar al menos un ataque. La pelea se desarrolla mediante ataques aleatorios por turno, hasta que alguno de los dos Pokémon queda con *hp* menor o igual a 0. La función debe devolver al ganador.

Ejercicio 7

Definir el objeto `ditto`. Este Pokémon tiene *hp* inicial 100, es de tipo normal y su único ataque `'ataqueCopiar'` consiste en copiar algún ataque del enemigo. Por ejemplo, después de ejecutar `ditto.ataqueCopiar(raichu)`, Ditto podría aprender `ataqueImpactrueno`.

Pautas de Entrega

Todo el código producido por ustedes debe estar en el archivo `taller.js` y es el **único** archivo que deben entregar. Para probar el código desarrollado abrir el archivo `TallerJS.html` en un navegador web. Cada función o método (incluso los auxiliares) asociado a los ejercicios debe contar con un conjunto de casos de test que muestren que exhibe la funcionalidad esperada. Para esto se deben modificar las funciones *testEjercicio* en el archivo fuente *taller.js*, agregando todos los tests que consideren necesarios (se incluyen algunos tests de ejemplo). Pueden utilizar la función auxiliar *res.write* para escribir en la salida. Si se le pasa un booleano como segundo argumento, el color de lo que escriban será verde o rojo en base al valor de dicho booleano. Se debe enviar un e-mail a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-POO]` seguido inmediatamente del nombre del grupo.
- El código JavaScript debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `taller.txt` (pueden cambiarle la extensión para que no sea detectado como posible software malicioso).
- El código entregado **debe** incluir tests que permitan probar las funciones definidas.

No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté **adecuadamente** comentado (son comentarios adecuados los que ayudan a entender lo que no es evidente o explican decisiones tomadas; no son adecuadas las traducciones al castellano del código). Los objetivos a evaluar son:

- Corrección.
- Declaratividad.
- Prolijidad: evitar repetir código innecesariamente y usar adecuadamente los métodos previamente definidos.

Importante: se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

Referencias del lenguaje JavaScript

Como principales referencias del lenguaje de programación JavaScript, mencionaremos:

- **W3Schools JavaScript Reference:** disponible online en: https://www.w3schools.com/jsref/default.asp.
- **MDN Web Docs: Mozilla - Referencia de JavaScript**: disponible online en: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia.