Trabajo Práctico 2
Programación Orientada a Objetos

Paradigmas de Lenguajes de Programación — 2do cuat. 2023

Fecha de entrega: 7 de noviembre de 2023

1. Introducción

La empresa Robots Argentinos S.A. nos encargó el diseño de su nueva línea de robots para uso doméstico. El plan para la primera etapa del desarrollo consistirá en modelar nuevos robots a partir de un prototipo original, llamado Malambo. Usando a Malambo como base, se podrán crear nuevos robots a su imagen y semejanza para luego darles instrucciones específicas respecto a la tarea que el nuevo robot tiene asignada.

Representación utilizada

Los robots son objetos que saben responder los siguientes mensajes:

1. ***nombre***: que retorna el texto correspondiente al nombre del robot
2. ***peso***: que retorna el peso en gramos del robot
3. ***directiva***: que retorna el texto correspondiente a la misión que tiene asignada el robot

2. Ejercicios

Ejercicio 1

Definir el objeto `Malambo`, que representa al robot Malambo. Malambo pesa 500 gramos y su directiva es *limpiar*.

Ejercicio 2

(a) Definir la función `nuevoRobot` que toma un nombre, un peso y una directiva y retorna un nuevo robot basado en Malambo pero con sus respectivos atributos actualizados.

(b) Utilizando la función `nuevoRobot` crear a Chacarera. Chacarera pesa 1500 gramos y su directiva es *cortar el pasto*.

Ejercicio 3

(a) Agregarle a Malambo el método `presentarse` que debe devolver el texto "*Hola, soy Malambo y me encanta limpiar.*" Asegurarse que todos los robots basados en Malambo puedan responder este mensaje correctamente (es decir, con su propio nombre y directiva).

(b) Agregarle a Malambo el método `limpiar` que no toma argumentos, incrementa en uno el peso de Malambo y retorna el texto "*limpiar.*"

Ejercicio 4

Así como Malambo sabe ejecutar el método `limpiar` (porque su directiva es *limpiar*), cada robot debería tener definida también una función particular relativa a su directiva.

Definir una **función constructora** `Robot` que además de inicializar un nuevo robot (con los mismos argumentos que `nuevoRobot`) tome un argumento adicional que será una función (relacionada a su directiva) que el nuevo robot deberá ejecutar cuando reciba el mensaje correspondiente a su directiva.

Es importante que los robots creados mediante esta función puedan presentarse, sin alterar la cadena de prototipos (es decir, sin cambiar el prototipo de ningún objeto, incluyendo el nuevo) ni repetir código.

Ejercicio 5

Uno de los problemas que enfrenta la empresa con respecto al entrenamiento de sus robots es que, cuando se desea enviar un mensaje a un robot específico, a veces resulta difícil encontrarlo. Crear al robot mensajero `Milonga` que pesa 1200 gramos y se encarga de entregar mensajes (en otras palabras, de *mensajear*). La función asociada a su directiva debe recibir como argumentos un remitente, un destinatario y el mensaje que el remitente le está mandando al destinatario. El sistema de mensajería funciona de la siguiente forma:

1. Si el destinatario sabe responder al mensaje del remitente, el mensajero debe entregar el mensaje al destinatario. En caso contrario, la función retorna el mensaje original.

2. El resultado de haber entregado el mensaje al destinatario también es un mensaje que el mensajero debe entregar como respuesta del destinatario al remitente.

3. Si el remitente sabe responder a la respuesta del destinatario, el mensajero debe entregar la respuesta al remitente. En tal caso, la función debe retornar el resultado de tal ejecución. En caso contrario, la función retorna la respuesta del destinatario.

Ejercicio 6

Luego del éxito de Milonga, la empresa decide crear una nueva línea de robots mensajeros.

Definir la función constructora `RobotMensajero` de manera tal que los robots construidos mediante esta función puedan enviar mensajes del mismo modo que Milonga (sin repetir código ni usar a Milonga como prototipo), además de tener las funcionalidades comunes a todos los robots. Se recomienda utilizar la función `call(receptor, argumentos)` que se encuentra definida en `Function.prototype` para inicializar los atributos comunes.

Dado que la forma de mensajear es común a todos los robots mensajeros, se espera que este comportamiento se defina de manera acorde y no se incluya explícitamente como atributo de cada robot.

Ejercicio 7

La empresa se dio cuenta que algunos trabajos requieren más esfuerzo que otros. Los robots que tienen las tareas más difíciles necesitan ayuda y los que tienen las tareas más fáciles terminan pronto y no tienen nada que hacer. Para que los robots ociosos puedan ayudar a los demás, se necesita que puedan modificar su directiva.

(a) Hacer lo necesario para que todos los robots (actuales y futuros) puedan responder al mensaje `reprogramar` que toma como argumento la nueva directiva a ser asignada y su función asociada. La directiva pasada como argumento debe ser diferente a la que ya tenía. En caso contrario, el robot se confunde y su nueva directiva pasa a ser "...".

(b) Hacer lo necesario para que todos los robots (actuales y futuros) puedan responder al mensaje `solicitarAyuda` que toma como argumento otro robot y se lo asigna como ayudante. Al ayudante se le debe asignar la misma directiva que la del robot que lo solicitó (y debe poder ejecutar su misma función asociada). Un robot que ya tiene ayudante no puede solicitar un segundo ayudante. Si esto pasara, el robot que solicitó ayuda debe indicarle a su ayudante que sea él quien solicite otro ayudante. Por ejemplo:

1. B solicita ayuda de A.
2. A pasa a ser ayudante de B.
3. B solicita ayuda a C pero como B ya tiene un ayudante, le pasa el pedido a A y C pasa a ser ayudante de A.

3. 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.

- El código de cada ejercicio debe escribirse dentro de la sección correspondiente. No se permite modificar el código de los ejercicios anteriores.

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.