Trabajo Práctico 2
Programación Orientada a Objetos

Paradigmas de Lenguajes de Programación — 1er cuat. 2021

Fecha de entrega: 10 de junio de 2021

1. Introducción

Las agencias de inteligencia Control y Kaos se encuentran compitiendo por la supremacía logística internacional. Se desea modelar en Javascript el comportamiento de los agentes de cada una de ellas. Tanto los agentes como las agencias se modelarán con **objetos**. Cada agencia tiene un **programa de entrenamiento** que le permite **crear** nuevos agentes. También utiliza **mensajes** específicos para que sus agentes puedan comunicarse entre sí. Un agente puede ser enviado a espiar a la agencia enemiga. Para eso debe simular ser un agente de la otra agencia, lo que implica conocer los **mensajes** que utilizan para comunicarse.

2. Ejercicios

Ejercicio 1

a) Definir la función constructora `AgenteDeControl` que permita crear un agente de la agencia Control. Por ahora, un agente de Control sólo deberá responder al mensaje `agencia` que devuelve la cadena "Control".

b) Utilizando la función constructora `AgenteDeControl` crear al agente `smart`.

Ejercicio 2

a) Definir la función constructora `Agencia` que toma como argumento otra función constructora (pero de agentes) y la utiliza para crear una nueva agencia cuyo programa de entrenamiento sea la función constructora de agentes pasada por parámetro.

b) Utilizando la función constructora `Agencia` crear a la agencia `control` que utiliza `AgenteDeControl` como programa de entrenamiento.

Ejercicio 3

a) Definir la función `nuevoAgente` que dada una Agencia crea y devuelve un nuevo agente de dicha agencia.

b) Definir la función `enrolar` que dado un agente (que no pertenece a ninguna agencia) y una agencia, enrola al agente en la agencia, haciéndolo pasar por su programa

de entrenamiento. **Sugerencia**: ver la función `bind` de `Function.prototype`: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Function/bind

Ejercicio 4

Cada agente dentro de una agencia se identifica con un número. Estos números se van asignando a los agentes de forma creciente. El primer agente de una agencia recibirá el número identificador 1. El siguiente recibirá el 2 y así sucesivamente.

Se desea que todos los agentes conozcan su número de identificación, así como la cantidad total de agentes en su agencia (o lo que es equivalente, el número del último agente asignado). Sin embargo, como son agencias secretas, cada agencia usa mensajes distintos para obtener esos datos. Se sabe por ejemplo que los agentes de Control utilizan los mensajes "idC" y "nC" para responder con su número de agente y con el número total de agentes, respectivamente. Por su parte, los agentes de Kaos usan los mensajes "idK" y "nK" en su lugar.

Modificar lo que sea necesario para que:

- la función constructora `Agencia` tome otros dos argumentos correspondientes a los identificadores de mensajes, de forma que `control` pueda definirse como el resultado de `new Agencia(AgenteDeControl, ''idC'', ''nC'')`,
- si un agente pertenece a una agencia (ya sea porque fue creado con `nuevoAgente` o enrolado con `enrolar`) sepa responder correctamente a los 2 mensajes correspondientes a su agencia (tener en cuenta que el valor de la cantidad total de agentes se va actualizando y los agentes deberían ser conscientes de esos cambios) y
- tanto `nuevoAgente` como `enrolar` sigan funcionando como se espera (ojo con el código repetido).

Ejercicio 5

Modificar lo que sea necesario para que todos los agentes que sean creados o enrolados a partir de ahora puedan ser enviados a espiar a través del mensaje `espiar` (que toma como argumento la agencia objetivo) y puedan dejar de espiar a través del mensaje `dejarDeEspiar` (que no toma argumentos y devuelve al agente a su agencia original).

Cuando un agente es enviado a espiar debe comportarse como si hubiera pasado por el programa de entrenamiento de la agencia a la que va a espiar. Debe aprender a responder el mensaje que corresponde a la cantidad total de agentes en la agencia objetivo (el cual también debe actualizarse) pero no se le debe asignar un número identificador sino que debe mantener el de su agencia original. Si a un agente en rol de espía se le pide que vaya a espiar este debe cumplir para no levantar sospechas y comportarse como un espía de la agencia que está espiando. Por lo tanto, un espía podría ser enviado a espiar a su propia agencia (suponer que una agencia no le va a pedir a un agente que se espíe a sí misma).

Cuando un espía deja de espiar vuelve a su agencia original, sin importar a cuál está espiando (es decir, aunque sea un doble espía). En definitiva, la solución implementada debería comportarse de la siguiente manera (suponiendo definidas las agencias `control` y `kaos`):

```
let a = nuevoAgente(control); // sabe responder idC y nC, pero no idK ni nK
a.espiar(kaos);               // sabe responder idC y nK, pero no idK ni nC
a.dejarDeEspiar();            // sabe responder idC y nC, pero no idK ni nK
```

```
let b = nuevoAgente(kaos);     // sabe responder idK y nK, pero no idC ni nC
b.espiar(control);             // sabe responder idK y nC, pero no idC ni nK
b.espiar(kaos);                // sabe responder idK y nK, pero no idC ni nC
b.dejarDeEspiar();             // sabe responder idK y nK, pero no idC ni nC
```

Ejercicio 6

a) Definir la función constructora `agenteEspecial` que dada una agencia y una habilidad especial (representada como una función) devuelve un nuevo agente de la agencia que además de comportarse como cualquier otro agente de esa agencia, posee la habilidad especial pasada como segundo argumento. El agente creado debe poder responder al nombre de la función que representa su habilidad y responder ejecutándola.

b) Definir la habilidad `camuflar` como una función que dado un objeto le otorga al agente que posee tal habilidad todos los atributos del objeto pasado como parámetro. Esta habilidad debe funcionar de la siguiente manera:

```
let camaleon = new agenteEspecial(Control, camuflar);
camaleon.camuflar({
  color: 'rojo',
  repetir: function(s) {return s;}
}); // sabe responder 'color' y 'repetir'
```

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.

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.