TAI: La objetización de las personas

Javier Ribal del Río

2025-12-20

Table of contents

1 Objetivo de la tarea

El objetivo de esta tarea es aplicar los conocimientos básicos de programación orientada a objetos en JavaScript, haciendo uso de las clases definidas en ES6. En particular, se evaluará el uso de:

- herencia entre clases
- encapsulación mediante propiedades privadas
- getters y setters usando las palabras clave `get` y `set`
- métodos estáticos
- validación de datos

Para ello se definirá una jerarquía de clases que represente un equipo de Hyperloop.

2 Entregar

Se debe entregar un único archivo de JavaScript (`.js`) que contenga la solución completa a la tarea.

3 Realización de la tarea

El archivo JavaScript debe estar claramente dividido en dos partes:

- la **implementación de clases**, donde se definirán todas las clases necesarias
- la **zona de uso**, donde se utilizarán las clases previamente definidas

3.1 Implementación de clases

Todas las clases deberán definirse utilizando la sintaxis de **clases ES6**, haciendo uso correcto de:

- `class`
- `extends`
- `super`
- propiedades privadas (usando `#`)
- getters y setters (`get` y `set`)
- métodos estáticos

3.1.1 Clase `Person`

Debe contener:

- propiedad privada `name`
- propiedad privada `dni`
- Se puede obtener tanto el nombre como el DNI, pero **solo se puede modificar el nombre**
- El constructor recibirá:
  - un nombre
  - un **NÚMERO** de DNI
- El constructor deberá calcular automáticamente la letra del DNI siguiendo el algoritmo oficial: https://www.interior.gob.es/opencms/es/servicios-al-ciudadano/tramites-y-gestiones/dni/calculo-del-digito-de-control-del-nif-nie/
- Métodos y accesos:
  - getter `name`
  - setter `name`
  - getter `dni`
  - método `getInfo()` que devuelva un string con el nombre y el DNI completo
- Si los datos introducidos no son válidos, se lanzará un `Error`

3.1.2 Clase `Member` (clase hija de `Person`)

Representa a un miembro del equipo de Hyperloop.

Debe contener:

- propiedad privada `department`
- propiedad privada `yearsExperience`

3.1.2.1 Departamentos válidos

Los únicos departamentos permitidos son:

- `"Managment"`
- `"Operaciones"`
- `"Avionics"`
- `"Electromagnetics"`
- `"Mechanics"`

No se permitirá ningún otro valor.

3.1.2.2 Requisitos adicionales

- La propiedad `department`:
  - se accederá mediante:
    - getter `department`
    - setter `department`
- El setter validará que el departamento sea válido
- `yearsExperience` debe ser un número mayor o igual que 0

3.1.2.3 Método estático obligatorio

La clase `Member` debe incluir el siguiente método estático:

- `isValidDepartment(department)`
  - devuelve `true` si el departamento es válido
  - devuelve `false` en caso contrario
  - deberá utilizarse tanto en el constructor como en el setter de `department`

3.1.2.4 Métodos públicos

- getter `yearsExperience`
- método `getInfo()`:
  - sobrescribe el método de `Person`
  - incluye el departamento y los años de experiencia

3.1.3 Clase `Engineer` (clase hija de `Member`)

Representa a un ingeniero del equipo.

Debe contener:

- propiedad privada `specialty`
- La especialidad será un string no vacío

3.1.3.1 Accesos y métodos

- getter `specialty`
- método `getInfo()`:
  - sobrescribe el método anterior
  - incluye la especialidad del ingeniero

3.1.4 Clase `Team`

Representa el equipo completo de Hyperloop.

Debe contener:

- propiedad privada `members`, que será un array de objetos `Member`

3.1.4.1 Métodos públicos

- `addMember(member)`
  - solo permite añadir objetos que sean instancia de `Member`
- `removeMemberByDni(dni)`
- `listMembers()`
  - muestra por consola la información de todos los miembros del equipo
- `countMembersByDepartment(department)`
  - devuelve cuántos miembros pertenecen a un determinado departamento

3.2 Zona de uso

En la zona de uso del archivo JavaScript se debe:

- Crear al menos:
  - 2 objetos `Engineer` de departamentos distintos
  - 1 objeto `Member` que no sea ingeniero
- Crear un objeto `Team`
- Añadir los miembros al equipo
- Utilizar explícitamente:
  - getters
  - setters
  - el método estático de validación
- Mostrar por consola:
  - el listado completo del equipo
  - el número de miembros por departamento
- Probar al menos un caso incorrecto:
  - por ejemplo, asignar un departamento no válido
  - o intentar añadir un objeto que no sea `Member`
- Demostrar que el error se gestiona correctamente

4 Apéndice: comprobaciones y errores en JavaScript (NECESARIO PARA LA TAREA)

En el desarrollo de la tarea pueden aparecer algunas comprobaciones y mecanismos de control de errores que **es necesario saber implementar por cuenta propia**. Su inclusión tiene como objetivo facilitar la robustez del código, y **forman parte de los contenidos evaluables de la tarea**.

4.1 Uso de `Error`

Cuando se detecta una situación incorrecta (por ejemplo, un valor no válido o un uso indebido de una clase), se puede lanzar un error utilizando:

```
throw new Error("Mensaje de error");
```

Esto indica claramente que algo no es correcto y detiene la ejecución normal del programa.

4.2 Uso de `instanceof`

El operador `instanceof` se utiliza para comprobar si un objeto ha sido creado a partir de una clase concreta.

```
objeto instanceof Clase
```

Devuelve `true` si el objeto pertenece a esa clase y `false` en caso contrario.

4.3 Uso de `typeof`

El operador `typeof` permite comprobar el tipo de datos simples como strings o números.

```
typeof variable
```

Ejemplos:

```
typeof "texto"   // "string"
typeof 10        // "number"
```

Se utiliza para validar los datos recibidos por constructores o setters. Para comprobar clases u objetos personalizados debe usarse `instanceof`, no `typeof`.