Tarea 013 – Javascript – Herencia de clases – Empleado

Los resultados de todas las tareas incorporarán, además del código fuente, los comentarios precisos y necesarios para su fácil compresión. No escatimes esfuerzos en comentar el código, es una buena práctica para aprender, además de ser muy útil para modificaciones o reutilizaciones futuras.

Sería muy buena práctica añadir el título y el enunciado del ejercicio (como comentarios) al principio del código fuente.

Añade documentos en formato Word con capturas de la salida por pantalla (al ejecutar la página) si consideras que queda más clara la resolución del ejercicio.

Deberás entregar dos versiones del resultado:

1. **Un único documento en Word sin comprimir**: que contenga todas las líneas de código, y por orden, primero el código **html**, luego el código **css** y por último el código **js**. Este archivo servirá para que el profesor haga anotaciones y/o correcciones a los alumnos.
2. **Los archivos de código fuente**: en un único archivo, ya sea de extensión: **html**, **js**, o **css**. Si precisas entregar varios archivos, comprímelos en un único **zip**. (No admito rar).

El nombre del archivo entregado comenzará por tu número, tu nombre y seguido por TareaXXX. Ejemplo: **00FedericoTarea009.zip**

013 – Ejercicio de Herencia de clases

Se pide crear unas clases con las propiedades y métodos necesarios para lograr el objetivo final.

Hay que tener en cuenta los siguientes aspectos:

- Habrá 3 clases: **Persona**, **Empleado** y **Cliente**. **Empleado** y **Cliente** son clases heredadas de **Persona**.
- Los objetos de la clase **Persona** tienen un identificador (atributo **id**) que será numérico, auto incremental, comenzando en 101.
- Los objetos de la clase **Empleado** tendrán un **id** que comenzará en 201.
- Los objetos de la clase **Cliente** tendrán un **id** que comenzará en 301.
- Todos los objetos creados pasarán por el constructor de la clase padre. Por lo que el atributo **id** de la clase **Persona** se verá incrementado al crear un nuevo objeto de cualquier clase.
- Todas las clases incorporan su método **toString**().
- Los nombres y apellidos, aunque se guarden en mayúsculas o minúsculas, se devolverán en formato “nombre propio”: Primera letra de cada palabra en mayúscula y las siguientes en minúscula.
- La fecha se devolverá en el formato local del navegador.
- El sueldo de los empleados se devolverá en formato de moneda local.
- La fecha de registro de los clientes será la fecha actual, aunque habrá método para cambiarla.
- Se podrán crear objetos en base a objetos de la clase persona como en la siguiente línea

```
let empleado2 = new Empleado ( persona1.nombre, persona1.apellido, persona1.edad, 30000 );
```

- Habrá una **constante** en la clase **Persona** que establezca la cantidad máxima de objetos de la clase **Persona**.
- La salida por consola de los métodos **toString**() será como en la figura de abajo. Para facilitar esta tarea se utilizará *Template String.*

Según los datos introducidos, en la consola del navegador debe verse algo similar a:

```
let persona1 = new Persona ('JUAN', 'PÉREZ', 19);
let empleado1 = new Empleado ('federica', 'lópez', 25, 22050.2);
let cliente1 = new Cliente ('AnTONio','García', 28, new Date());
console.log (persona1.toString());
console.log (empleado1.toString());
console.log (cliente1.toString());
let empleado2 = new Empleado ( persona1.nombre, persona1.apellido, persona1.edad, 30000 );
console.log (empleado2.toString());
let persona2 = new Persona ('Alejandro', 'Muñoz', 30);
console.log (persona2.toString());
console.log ("Id del Empleado2: "+empleado2.idEmpleado);
console.log ("id del Cliente 1: "+cliente1.idCliente);
console.log ('Contador de Personas: '+ Persona.contadorPersonas);
let p23=new Persona();
console.log ('Contador de Personas: '+ Persona.contadorPersonas);
let p24=new Persona();
console.log ('Contador de Personas: '+ Persona.contadorPersonas);
let p25=new Persona();
console.log ('Contador de Personas: '+ Persona.contadorPersonas);
let p26=new Persona();
console.log ('Contador de Personas: '+ Persona.contadorPersonas);
```

```
101:                                                    persona_herencia.js:93
        Juan Pérez
        Edad:19
Empleado 201:                                           persona_herencia.js:94
        Federica López
        Edad:25,
        Sueldo: 22.050,20 €
Cliente 301:                                            persona_herencia.js:95
        Antonio García
        Edad:28,
        Registro: 11/12/2022
Empleado 202:                                           persona_herencia.js:97
        Juan Pérez
        Edad:19,
        Sueldo: 30.000,00 €
105:                                                    persona_herencia.js:99
        Alejandro Muñoz
        Edad:30
Id del Empleado2: 202                                   persona_herencia.js:100
id del Cliente 1: 301                                   persona_herencia.js:101
Contador de Personas: 105                               persona_herencia.js:102
Contador de Personas: 106                               persona_herencia.js:104
Contador de Personas: 107                               persona_herencia.js:106
AVISO: Hay más de 6 objetos creados.                    persona_herencia.js:8
Contador de Personas: 108                               persona_herencia.js:108
AVISO: Hay más de 6 objetos creados.                    persona_herencia.js:8
Contador de Personas: 109                               persona_herencia.js:110
```

Para conseguirlo debes realizar lo que se pide a continuación:

1º Diseñar la estructura de clases necesaria.

2º Crear las clases.

3º Hacer las modificaciones que permitan devolver los valores en los formatos que se piden.

4º Establecer una constante con el valor máximo de objetos tipo Persona. A partir de este valor, se verá un AVISO en consola.

Consejos

En el constructor de la clase Persona, utilizando el atributo estático **contadorPersonas** hay que establecer un identificador único al atributo **idPersona** y también un identificador único para la clase Empleado y la clase Cliente.

idEmpleado e **idCliente** tendrán tener sus propios contadores estáticos (en un caso real parece lo más conveniente),

Los métodos **toString**(), de las clases **Empleado** y **Cliente**, deberán utilizar la sobrecarga y/o polimorfismo del método **toString**().

Ayuda: Formato moneda, punto separador de miles y decimales:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Number/toLocaleString

```
var number = 123456.789;

// solicitar un formato de moneda
console.log(number.toLocaleString('de-DE', { style: 'currency', currency: 'EUR' }));
// → 123.456,79 €

// en Japones yen no utiliza una moneda menor
console.log(number.toLocaleString('ja-JP', { style: 'currency', currency: 'JPY' }))
// → ¥123,457

// limitar a tres digitos el significante
console.log(number.toLocaleString('en-IN', { maximumSignificantDigits: 3 }));
// → 1,23,000

// Utilizar el lenguaje por defecto del host con opciones para el formato del número
var num = 30000.65;
console.log(num.toLocaleString(undefined, {minimumFractionDigits: 2, maximumFractionDigits: 2}))
// → "30,000.65" donde English es el lenguaje por defecto, o
// → "30.000,65" donde Aleman es el lenguaje por defecto, o
// → "30 000,65" donde French es el lenguaje por defecto
```