Práctica 3

Clases y Excepciones

## Objetivos:

* Entender:
  + POO en JS
  + Utilizar y Entender las excepciones y su manejo en JS.

## Referencias:

* [Clases en javascript](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Classes)
* [Clases en typescript](https://www.typescriptlang.org/docs/handbook/classes.html)
* [throw](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling#throw_statement) y [try..catch](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling#try...catch_statement)

## 1 Objetos y Clases

**Nota 1:**

Si bien las relaciones y el protocolo son bastante evidentes a partir del enunciado cuando se pide que diseñe considere agregar protocolo o relaciones de conocimiento entre objetos que sea necesarias para su solución.

**Nota 2:**

Para los alumnos que vayan a trabajar con typescript en las entregas, les recomendamos que implementen estos ejercicios utilizando typescript

### **Ejercicio 1:**

En una red social los usuarios hacen posts. En este primer ejercicio nos interesa modelar solo eso. De los usuarios se conoce:

* Nombre y apellido
* Nickname
* Fecha de nacimiento (puede utilizar objetos Date, ya definidos en JS).

Cada usuario puede hacer posts, y los conoce, es decir, a un usuario le podemos pedir todos sus posts. Es razonable también pensar que a un post se le puede preguntar el autor del mismo.

De un post se conoce:

* Fecha y hora (idem anterior)
* Contenido: un string

El protocolo para esta operación podría ser algo como:

aUser.post('Que lindo es JS');

donde aUser es el usuario que postea, y el objeto post se crea a partir del string pasado como parámetro

Tareas:

1. Diseñe (realice un diagrama de clases UML)
2. Implemente utilizando clases ES6.

### **Ejercicio 2:**

En la misma red social ahora se agrega la posibilidad de tener amigos (instancias de la clase Usuario). Un usuario debe poder agregar o sacar usuarios a su lista de amigos.

El hecho de tener amigos implica que cuando un usuario hace un post todos sus amigos se enteran, es decir, reciben un mensaje que les informa que alguien hizo un post.

Un posible protocolo para esta notificación sería:

user.postDoneBy(userWhoPosted, aPost)

Donde: user es el usuario que se está enterando del post. aPost es lo que alguien posteó, aUser es quien ha posteado.

Cuando un usuario postea algo debe mostrarse un mensaje en la consola: quien lo hizo, qué posteó y cuando.

Cuando un usuario se entera que un amigo posteó algo, debe mostrarse en la consola que recibió esa notificación, cuál fue el post y quien lo hizo.

Tareas:

1. Defina los cambios necesarios, defina el protocolo necesario para los objetos y diseñe (realice un diagram de clases UML)
2. Implemente utilizando clases ES6.

### **Ejercicio 3:**

Nuestra red social sigue creciendo y ahora debe ser posible comentar los posts. Los comentarios son como posts: tienen un texto, pero con la limitación de solo poder tener 100 caracteres. Los comentarios conocen el post al cual refieren, también conocen la fecha, hora y autor del comentario. Un post puede tener muchos comentarios.

Los comentarios no pueden tener comentarios ni respuestas.

Cuando un usuario comenta un post de alguien, el “alguien” debe ser notificado. Un protocolo sugerido para esto sería:

aUser.postHasBeenCommentedBy(post, comment)

Donde aUser es el autor del post original, quien está siendo notificado. Comment es el comentario que se está notificando. Note que al comentario le podemos preguntar el autor para saber quien comentó.

Cuando un usuario hace un comentario debe imprimirse a la consola.

De la misma manera, cuando un usuario se entera que un post suyo ha sido comentado, debe imprimirse.

### **Ejercicio Opcional:**

Si le interesa entender cómo se puede implementar POO utilizando solamente prototipos y Funciones constructoras en Javascript puede leer:

* POO, Herencia, constructores y prototipos:
  + [Parte 1](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS)
  + [Parte 2](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes)
  + [Parte 3](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance)
* [Herencia y la cadena de prototipos](https://developer.mozilla.org/es/docs/Web/JavaScript/Herencia_y_la_cadena_de_protipos)
* [Objeto this en javascript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this)

## 

## 

## 2 Excepciones

1. Dada la clase Persona como de define a continuación.

|  |
| --- |
| class Persona {  constructor(\_nombre) {  this.nombre = \_nombre;  }  saludar(){  console.log(`Mucho gusto, soy ${this.nombre}`);  } } pepe = new Persona('Pepe'); pepe.saludar(); |

1. Envíele un mensaje que no entienda, por ejemplo persona.ladrar();
2. Qué imprime en la consola?
3. Ahora maneje la excepción, el comportamiento esperado de su programa es que se reporte por consola que se le ha envíado un mensaje a un objeto y que este no lo entendió
4. Implemente la función dividirPor(numerador, denominador). Para el caso en que el denominador sea 0 debe levantar una una excepción llamada: ZeroDivideError, en el mensaje se debe indicar cual era numerador que se quiso dividir por 0. **Siempre**, al terminar la ejecución de la función se debe imprimir en el log un mensaje indicando los parámetros de la función.

3. En el ejercicio de la red social, lance una excepción en los siguientes casos:

1. Cuando se intente crear un comentario con mas de 100 carácteres
2. Cuando si intente agregar como amigo a un usuario que ya es amigo.