****

# Recuperación de Examen final de Programación Imperativa

¡Llegó el momento de poner a prueba todo lo que estuvimos viendo a lo largo de estas semanas!

# Metodología de evaluación

Se evaluarán los siguientes conceptos sobre el código entregado:

* **FORMA**
  + Que el código esté prolijo e implemente buenas prácticas
  + Que las variables, métodos y funciones tengan nombres descriptivos
  + Que utilices nombres en español o en inglés pero no ambos
  + Que utilices camelCase donde corresponda
* **LÓGICA**
  + Que la lógica corresponda con lo que solicitan las consignas
  + Que utilices los métodos más adecuados para cada caso
* **FUNCIONAMIENTO**
  + Que el código funcione correctamente, sin arrojar errores
  + Que el código produzca el resultado esperado a partir de los datos suministrados

# Duración, formato y entrega

El examen tendrá una duración de 100 minutos (aprox). La hora de finalización será informada por el docente. ⚠️**Las entregas realizadas después del tiempo estipulado no serán tenidas en cuenta**.  
  
Al terminar el examen, deben entregarlo haciendo uso del formulario que les enviará el docente por chat de zoom.

⚠️**Recuerden verificar la recepción del formulario con el/la docente antes de retirarse del zoom**⚠️

# Algunos consejos 🤓👌

* Es normal tener nervios en esta etapa. Recordá que practicamos mucho estas semanas y tratá de mantener la mayor calma que puedas.
* Lee todas las consignas antes de empezar el examen para tener una idea general de lo que se pide.
* Si alguna consigna no queda clara, recordá que podés pedir aclaración por privado al docente.
* Si una consigna te bloquea demasiado, pasá a la siguiente, lo más probable es que luego se te ocurra una solución.



# Introducción

En este examen estaremos modelando funcionalidades para un sistema de tickets de mesa de ayuda del área de soporte IT de una empresa, teniendo en cuenta las características de los mismos en base a determinadas condiciones y generando la lógica necesaria para simular el proceso de una mesa de ayuda.

# Consignas

### **Preparando el entorno...**

Descarga [esta carpeta](https://drive.google.com/drive/folders/1TfkBBf_8ZaDTxwa9HusnHyxmye4etW6j?usp=sharing) con los archivos necesarios, encontrarás un .json con los datos de los tickets, un módulo jsonHelper.js con los métodos para leer y escribir en json y una plantilla donde desarrollarás las consignas, con esto deberás:

1. Crear un objeto literal que represente la aplicación.   
   El objeto será la representación de nuestro sistema de gestión de tickets, podemos llamarlo **mesaDeAyuda** contendrá todas las propiedades y métodos necesarios.
2. Agregar una propiedad llamada ***tickets*** en la que asignarás los tickets obtenidos a partir del método *leer* del objeto requerido como módulo (jsonHelper.js), el cual debes requerir como cualquier módulo, al comienzo del archivo, como hemos visto en las prácticas previas.
3. Agregar un método **listarTickets** que reciba como parámetro un array de tickets y los imprima por consola.  
   Este método deberá imprimir por consola un mensaje con el siguiente formato:

*Id: {id}, de {usuario}, {resuelto}* **,** *espera de {minutosDeEspera} minutos*

**Ejemplos:**

*Id: 1, de Regina Cejas, está resuelto, espera de 175 minutos*

*Id: 6, de Carmela Newman, no está resuelto, espera de 175 minutos*

⚠️ **Importante:** Este método listarTickets deberá ser invocado en cada ejecución de todos aquellos métodos que retornan un array de tickets. Digamos que reemplazan al console.log() para hacer más agradable el muestreo de datos.

**Resultado esperado al ejecutar el método:** un mensaje por consola por cada ticket con el formato indicado.

1. Agregar un método **buscarPorId** que permita buscar un ticket en función de su id.
   * Este método recibirá por parámetro un valor de tipo **Number** que represente el **id** a buscar.
   * En caso de encontrar un ticket, devolverá el objeto literal que lo representa.
   * En caso contrario devolverá *undefined*
2. Agregar un método **ticketsResueltos**,que retorne todos los tickets resueltos, es decir, aquellos que tengan la propiedad ***estaResuelto***en **true**.
   * Este método no recibirá ningún parámetro.
   * Este método debe recorrer el listado de tickets completo para filtrar.
   * Este método retornará un array con los tickets resueltos.
3. Agregar un método **filtrarPorEspera** que permita filtrar los tickets que tengan una duración que esté entre el mínimo y máximo enviado.
   * Este método recibirá por parámetro dos valores de tipo **Number** que representan la duración mínima y máxima de tiempo de espera para buscar.
   * Este método devolverá un array con todos los tickets que tengan una duración mayor o igual al primer parámetro y menor o igual al segundo parámetro.
   * En caso de no encontrar ningún ticket, devolverá un array vacío.
4. Agregar un método **ordenarPorEspera** que permita ordenar todos los tickets de mayor a menor según su propiedad ***minutosDeEspera***.
   * Este método no recibirá ningún parámetro
   * Este método retornará un array con todos los tickets ordenados por su tiempo de espera.
5. Agregar un método **duracionTotal** que permita calcular la duración total en minutos de todos los tickets.
   * Este método no recibirá ningún parámetro.
   * Este método retornará un string con el siguiente formato:

*La duración de todos los tickets sumadas es de {minutosDeEspera} minutos*

1. Agregar un método **cambiarPrioridad** que permita cambiar la prioridad de un ticket en función de su id y guardar los cambios en el archivo JSON.  
   ***⚠️ Importante:*** *el método de escritura modifica el archivo JSON original, te recomendamos tener una copia a mano por si algo sale mal y necesitás restaurar el archivo.*
   * Este método recibirá por parámetro un valor de tipo **Number** que representa el **id** a buscar y un valor tipo **string** representando la nueva prioridad
   * Este método utilizará el método **buscarPorId**
   * En caso de encontrar una ticket con el **id** buscado:
     1. Cambiará el valor de la propiedad **prioridad** con aquel recibido por parámetro.
     2. Escribirá los cambios en el archivo JSON que contiene los tickets.
   * En caso contrario devolverá *undefined*
   * Para verificar que el guardado fue correcto, en la sección de ejecuciones de este punto, debes volver a invocar el método que busca por título y que se muestre en consola que dicho ticket fue modificado correctamente y guardado correctamente en el json actualizándolo.

**⚠️ Importante:** el método que te damos para escribir el archivo JSON reemplaza todo el archivo, así que le vas a tener que pasar el array completo de tickets y no solo el elemento modificado 😉.