****

**Programación Imperativa**

# Segundo Parcial

## Objetivo

Integrar los conocimientos adquiridos hasta el momento en la cursada.  
Practicar la lectura, comprensión e interpretación de consignas y producir código que resuelva los enunciados propuestos.

## **Buenas prácticas**

A la hora de la resolución no solo es importante que el código funcione correctamente, sino también que sea legible y respete las buenas formas.

* Que las variables, métodos y funciones tengan nombres descriptivos
* Que se utilicen nombres en español o en inglés pero no ambos
* Que se utilice camelCase donde corresponda
* Que se utilicen los métodos más adecuados para resolver cada problema
* Que el código produzca el resultado esperado a partir de los datos suministrados

## Introducción

Una empresa que reúne diversos trabajadores de oficios necesita crear un pequeño sistema que le permita facilitar la gestión de éstos.

La empresa tiene una base de información que podremos encontrar en un archivo con formato JSON que contendrá un array de objetos literales que representan las propiedades de cada profesional.

### **Para comenzar:**

* Te damos una [carpeta .zip](https://drive.google.com/file/d/1KaT5zQ_elo_Dy3SIcsGeQuIQNARU9yoA/view?usp=sharing) (contiene 3 archivos) que debes descargar.
* Encontrarás algo de código ya escrito donde se requiere el módulo de jsonHelper que es el responsable de leer y guardar los cambios que sucedan en los datos de la cartilla de profesionales.
* También un objeto literal que representa a la empresa con algo de código ya escrito.
* Finalmente una estructura de logs e invocaciones de métodos que debes respetar indicando qué consigna está siendo ejecutada.

## Comprobaciones previas:

### **1. Ejecutar**

Comprobar que al ejecutar app.js se imprime en la consola el listado de trabajadores.

### **2. Analizar el objeto literal que representa a la empresa**

Este objeto literal contiene una **propiedad** con el listado de trabajadores obtenido en la línea 2 y algunas **funcionalidades** básicas que nos ayudarán a comprobar que los nuevos métodos solicitados están funcionando correctamente. Mira dentro de ellos e intenta comprender qué y cómo lo hacen aún cuando no lo logres plenamente.

## Consignas:

1. Vemos aquí una propiedad que contiene toda nuestra “base de datos”.   
   Responder en el código cuál es esa propiedad con un comentario.
2. También vemos algunas funciones. Responda con qué terminología se las denomina.
3. Vemos en el código que invocamos a un método (*catalogoTrabajadores.listarTodos*) del objeto que lista a todos los profesionales. Completa este punto 3 utilizando el método ya dado que muestra **un solo profesional**.
4. Agregar (crear) un método **filtrarPorRating** que permita filtrar trabajadores que estén por encima de un rating indicado, es decir, aquellos cuya propiedad *rating* sea mayor o igual a una dada.
   1. Este método deberá recibir un parámetro, el rating mínimo deseado.
   2. Este método deberá retornar un nuevo array de trabajadores cuyo rating sea mayor o igual al ingresado por parámetro.
   3. Ese array retornado puedes guardarlo en una nueva variable y utilizar el método ya dado de listarTodos pasándole por argumento dicho array.
5. Agregar un método **buscarPorId** que permite buscar dentro de la propiedad de nuestro objeto, a un profesional por la propiedad *id*, la misma deberá ser igual a un id que se envía como argumento al momento de invocarlo.
   * Este método deberá retornar un trabajador.
   * Debe mostrarse por consola utilizando el método ya dado que muestra **un solo trabajador**.
6. Agregar un método **incrementarConsultasRealizadas** que permite cambiar la propiedad *consultasRealizadas* incrementando el valor que ya trae la propiedad, él mismo deberá realizar los siguiente pasos.
   1. Recibe dos parámetros, el id del profesional y la cantidad de consultas que se desea sumar a las ya realizadas.
   2. Reutilizar el método **buscarPorId** para obtener al trabajador.
   3. Una vez encontrado al trabajador deberá cambiar el valor de la propiedad *consultasRealizadas*.
   4. Utilizando la función *escribirJson()* del módulo ya dado de lecturaEscritura tenemos que sobreescribir el archivo JSON de profesionales para persistir los datos modificados.
   5. Al finalizar deberá retornar al profesional en cuestión.