Nombre: Alison Melysa Pérez Blanco
Carné: 202400023
Sección: B -

SIMULADOR DE CALLCENTER

MANUAL TÉCNICO

Curso: Lenguajes Formales y de Programación
Ingeniero: Zulma Karina Aguirre Ordoñez
Auxiliar: River Anderson Ismalej Roman

- **Introducción**

Este documento describe la implementación y el funcionamiento del sistema de simulación de CallCenter desarrollado en JavaScript con Node.js. El sistema permite cargar registros de llamadas desde archivos CSV, procesar la información de operadores y clientes, clasificar el nivel de satisfacción de las llamadas y generar reportes en formato HTML.

Se detallan las principales clases, funciones, estructura de carpetas, flujo de ejecución y componentes del sistema, con el fin de facilitar su comprensión, mantenimiento y futuras mejoras.

- **Objetivos**

El objetivo principal de este manual es:

- Proporcionar una guía clara y completa del funcionamiento interno del sistema.
- Facilitar la comprensión del código a nuevos desarrolladores, auxiliares o estudiantes.
- Documentar la arquitectura modular del sistema para permitir su extensión o modificación.
- Servir como referencia técnica para el análisis, depuración y mejora del sistema.

- **Dirigido**

Este manual está dirigido a estudiantes y desarrolladores interesados en entender y modificar el código de "Simulador de CallCenter".

- **Requerimientos**

Para ejecutar y modificar el sistema, se requiere lo siguiente:

Hardware

Computadora de escritorio o portátil.

Mínimo 4 GB de Memoria RAM.

Al menos 50 GB de espacio en disco (recomendado SSD).

Software

Windows 10 o superior, macOS o Linux.

Node.js v14 o superior (recomendado v18 o v20 LTS).

Editor de código

Visual Studio Code (recomendado) o cualquier editor moderno (Sublime Text, WebStorm, etc.).

Dependencias

Módulos nativos de Node.js: fs, path, readline (no requieren instalación adicional).

No se requiere npm install si el proyecto no tiene package.json (es un script autónomo).

Entradas del sistema

Archivos CSV con formato válido ubicados en la carpeta Data/.

Estructura esperada del CSV:

id_operador,nombre_operador,estrellas,id_cliente,nombre_cliente

4,Carlos,x;x;x;0;0,202,Agustina

- **Estructura del Proyecto**

El sistema está organizado en una estructura modular y clara:

- **Modelos**
  - CallRecord.js
  - Cliente.js
  - Operador.js
- **Servicios**
  - CallCenter.js
  - Menu.js
  - Reportes.js
- **Reportes (Carpeta generada automáticamente)**
  - Historial_Llamadas.html
  - Listado_Operadores.html
  - Listado_Clientes.html
  - Rendimiento_Operadores.html
- Data
  - ArchivoN.csv (Ejemplo de archivo de entrada)

index.js

MODELOS

CallRecord.js

- **Clase CallRecord**
  - **Constructor de CallRecord**

Crea una instancia que representa una llamada atendida por un operador.

- **Atributos almacenados**
  - **idOperador, nombreOperador:** identificación del operador.
  - **estrellas:** número entero (0 a 5) que representa la calificación.
  - **idCliente, nombreCliente:** identificación del cliente.
  - **rawStarsStr:** cadena original de calificación (ej: "x;x; ;x;").

  rawStarsStr || "".padEnd(5, "0"): si no se proporciona esta cadena, se usa "00000" como valor por defecto (5 ceros).

- **Método getClasificacion()**

Clasifica la llamada según el número de estrellas.

Reglas:

4 o 5 estrellas → "Buena"

2 o 3 estrellas → "Media"

0 o 1 estrella → "Mala"

Uso: Se usa en reportes y estadísticas.

- **Exportación del módulo**

Permite que otras partes del sistema (como CallCenter.js) puedan importar y usar esta clase.

```
class CallRecord {
    constructor(idOperador, nombreOperador, estrellas, idCliente, nombreCliente, rawStarsStr) {
        this.idOperador = idOperador;
        this.nombreOperador = nombreOperador;
        this.estrellas = estrellas;
        this.idCliente = idCliente;
        this.nombreCliente = nombreCliente;
        this.rawStarsStr = rawStarsStr || "".padEnd(5, "0");
    }

    getClasificacion() {
        if (this.estrellas >= 4) return "Buena";
        if (this.estrellas >= 2) return "Media";
        return "Mala";
    }
}

module.exports = CallRecord;

```

- **Cliente.js**

➢ **Clase Cliente**

Representa a un cliente del sistema.

- **Atributos**
  - **id:** identificador único.
  - **nombre:** nombre completo.
- **Exportación**

Permite que CallCenter.js cree instancias de Cliente al procesar el CSV.

```
class Cliente {
    constructor(id, nombre) {
        this.id = id;
        this.nombre = nombre;
    }
}

module.exports = Cliente;

```

- **Operador.js**

➢ **Clase Operador**

- **Constructor de Operador**

Crea un operador con un historial vacío de llamadas.

- **llamadas = []:** array que almacenará todas las CallRecord que atendió.

```
class Operador {
    constructor(id, nombre) {
        this.id = id;
        this.nombre = nombre;
        this.llamadas = [];
    }

```

- **Método agregarLlamada**

Añade una llamada al historial del operador.

Cuando se procesa una línea del CSV, se agrega la llamada al operador correspondiente.

- **Método cantidadLlamadas**

Devuelve cuántas llamadas ha atendido el operador.

Se usa en el reporte de rendimiento para calcular porcentajes.

- **Exportación**

Permite que CallCenter.js cree y gestione operadores.

```
8       agregarLlamada(llamada) {
9           this.llamadas.push(llamada);
10      }
11
12      cantidadLlamadas() {
13          return this.llamadas.length;
14      }
15  }
16
17  module.exports = Operador;
18
```

SERVICIOS

CallCenter.js

- **Importaciones**
  - **fs y path:** Módulos nativos de Node.js para leer archivos y manejar rutas.
  - **RegistroLlamada:** clase CallRecord (renombrada localmente).
  - **Operador y Cliente:** clases de modelos.

```
1   const fs = require("fs");
2   const path = require("path");
3   const CallRecord = require("../Modelos/CallRecord");
4   const Operador = require("../Modelos/Operador");
5   const Cliente = require("../Modelos/Cliente");
```

➢ **Clase CallCenter**

- **Constructor de CallCenter**
  - **operadores:** Map(id → Operador) → acceso rápido por ID.
  - **clientes:** igual, para evitar duplicados.
  - **llamadas:** lista de todas las llamadas cargadas.
  - **archivoCargado:** guarda la ruta del último archivo cargado (usado en reportes).

```
7   class CallCenter {
8       constructor() {
9           this.operadores = new Map();
10          this.clientes = new Map();
11          this.llamadas = [];
12          this.archivoCargado = null;
13      }
```

- **Método limpiarDatos**

Restablece el sistema a estado vacío.

Se usa cuando el usuario elige la opción 9 del menú.

```
15      limpiarDatos() {
16          this.operadores.clear();
17          this.clientes.clear();
18          this.llamadas = [];
19          this.archivoCargado = null;
20      }
```

- **Método parseLine**

Convierte una línea de texto (CSV) en un objeto CallRecord.

Pasos:

- Divide por , y elimina espacios.
- Verifica que haya al menos 5 campos.
- Extrae los datos.
- Procesa estrellasStr:
  - Divide por ;.
  - Limpia y convierte a minúsculas.
  - Cuenta cuántas veces aparece "x".
  - Limita el resultado entre 0 y 5 (Math.max/min).
- Crea y devuelve una nueva CallRecord.

Devuelve null si la línea es inválida.

```
22      parseLine(line) {
23          const parts = line.split(",").map(p => p.trim());
24          if (parts.length < 5) return null;
25
26          const idOperador = parts[0];
27          const nombreOperador = parts[1];
28          const estrellasStr = parts[2];
29          const idCliente = parts[3];
30          const nombreCliente = parts[4];
31
32          const estrellasArray = estrellasStr.split(";").map(e => e.trim().toLowerCase());
33          const countX = estrellasArray.filter(e => e === "x").length;
34          const estrellas = Math.max(0, Math.min(5, countX));
35
36          return new RegistroLlamada(idOperador, nombreOperador, estrellas, idCliente, nombreCliente, estrellasStr);
37      }
```

- **Inicio de cargarArchivo**
  - **fs.readFileSync:** lee el archivo completo (síncrono).
  - **split(/\r?\n/):** divide en líneas, compatible con Windows y Unix.
  - **trim() y filter():** elimina líneas vacías.

Si la primera línea empieza con "id_operador": se asume que es encabezado → se elimina.

Si no hay líneas válidas: muestra mensaje y termina.

```
39    cargarArchivo(ruta) {
40
41        try {
42            const contenido = fs.readFileSync(ruta, "utf8");
43            let lineas = contenido.split(/\r?\n/).map(l => l.trim()).filter(l => l.length > 0);
44
45            if (lineas[0]?.toLowerCase().startsWith("id_operador")) {
46                lineas = lineas.slice(1);
47            }
48
49            if (lineas.length === 0) {
50                console.log("El archivo está vacío.");
51                return;
52            }
```

- **Procesamiento de líneas**
  - **Bucle for:** recorre cada línea del archivo.
  - **parseLine:** intenta crear una llamada.
  - **Si falla:** muestra advertencia y sigue.
  - Si tiene éxito:
    - Se añade a this.llamadas.
    - Si el operador no existe, se crea uno nuevo.
    - Se agrega la llamada al operador.
    - Lo mismo para el cliente (si no existe, se crea).
  - **archivoCargado = ruta:** registra la fuente de los datos.
  - **Mensaje final:** informa cuántas llamadas se procesaron.
  - **try/catch:** maneja errores de lectura (archivo no existe, sin permisos, etc.).

```
54            for (let linea of lineas) {
55                const llamada = this.parseLine(linea);
56                if (!llamada) {
57                    console.warn("Línea con formato inválido:", linea);
58                    continue;
59                }
60                this.llamadas.push(llamada);
61
62                if (!this.operadores.has(llamada.idOperador)) {
63                    this.operadores.set(llamada.idOperador, new Operador(llamada.idOperador, llamada.nombreOperador));
64                }
65                this.operadores.get(llamada.idOperador).agregarLlamada(llamada);
66
67                if (!this.clientes.has(llamada.idCliente)) {
68                    this.clientes.set(llamada.idCliente, new Cliente(llamada.idCliente, llamada.nombreCliente));
69                }
70            }
71            this.archivoCargado = ruta;
72            console.log(`Archivo "${ruta}" cargado. Registros procesados: ${this.llamadas.length}`);
73        } catch (err) {
74            console.error("Error al leer el archivo:", err.message);
75        }
76    }
```

- **mostrarPorcentajeClasificacion**
  - Muestra en consola el % de llamadas por clasificación.
  - Si no hay llamadas: avisa.
  - Cuenta cuántas son "Buena", "Media", "Mala" usando getClasificacion().
  - Imprime el porcentaje con 2 decimales.

```
78      mostrarPorcentajeClasificacion() {
79          const total = this.llamadas.length;
80          if (total === 0) {
81              console.log("No hay llamadas cargadas.");
82              return;
83          }
84          const conteo = { Buena: 0, Media: 0, Mala: 0 };
85          for (let l of this.llamadas) {
86              conteo[l.getClasificacion()]++;
87          }
88          console.log("\n--------- Porcentaje de Clasificación de Llamadas ---------");
89          for (let tipo of ["Buena", "Media", "Mala"]) {
90              const pct = (conteo[tipo] / total) * 100;
91              console.log(`${tipo}: ${conteo[tipo]} llamadas — ${pct.toFixed(2)}%`);
92          }
93      }
```

- **mostrarCantidadPorEstrellas**
  - Muestra cuántas llamadas tienen 0, 1, ..., 5 estrellas.
  - **count:** objeto para contar por número de estrellas.
  - Imprime de 1 a 5 estrellas, y luego 0 por separado.

```
95      mostrarCantidadPorEstrellas() {
96          if (this.llamadas.length === 0) {
97              console.log("No hay llamadas cargadas.");
98              return;
99          }
100         const count = {0:0,1:0,2:0,3:0,4:0,5:0};
101         for (let l of this.llamadas) {
102             count[l.estrellas]++;
103         }
104         console.log("\n--------- Cantidad de llamadas por calificación (1..5) ---------");
105         for (let i = 1; i <= 5; i++) {
106             console.log(`${i} estrella(s): ${count[i]}`);
107         }
108         console.log(`0 estrellas: ${count[0]}`);
109     }
```

- **Getters**

Métodos de acceso para que otras clases (como Reportes) puedan obtener datos sin acceder directamente a las propiedades.

Reportes.js los usa para generar los reportes.

- **Exportación**

Permite que Menu.js cree una instancia del centro de llamadas.

```
112     getLlamadas() { return this.llamadas; }
113     getOperadores() { return this.operadores; }
114     getClientes() { return this.clientes; }
115     getTotalLlamadas() { return this.llamadas.length; }
116     getarchivoCargado() { return this.archivoCargado; }
117 }
118
119 module.exports = CallCenter;
```

Menu.js

- **Importaciones**
  - **readline:** permite leer entrada del usuario en consola.
  - **CallCenter y Reportes:** servicios principales del sistema.

```
1  const readline = require("readline");
2  const CallCenter = require("./CallCenter");
3  const Reportes = require("./Reportes");
```

➢ **Clase Menu**

- **Constructor de Menu**
  - **callCenter:** instancia del sistema de datos.
  - **reportes:** instancia que genera reportes, usando el centro.
  - **rl:** interfaz para preguntar al usuario.

```
5  class Menu {
6      constructor() {
7          this.callCenter = new CallCenter();
8          this.reportes = new Reportes(this.callCenter);
9          this.rl = readline.createInterface({
10             input: process.stdin,
11             output: process.stdout
12         });
13     }
```

- **Iniciar()**

Inicia la aplicación mostrando el título y el menú.

```
15      Iniciar() {
16          console.log("======================================================================");
17          console.log("Simulador de CallCenter – JavaScript (Consola)");
18          this.mostrarMenu();
19      }
```

- **mostrarMenu()**

Muestra el menú con todas las opciones.

- **rl.question:** espera una entrada del usuario.
- **opt.trim():** elimina espacios antes y después.

```
21      mostrarMenu() {
22          console.log("\n======================== MENÚ PRINCIPAL ========================");
23          console.log("1. Cargar registros de llamadas");
24          console.log("2. Exportar Historial de Llamadas");
25          console.log("3. Exportar Listado de Operadores");
26          console.log("4. Exportar Listado de Clientes");
27          console.log("5. Exportar Rendimiento de Operadores");
28          console.log("6. Exportar TODOS los reportes");
29          console.log("7. Mostrar Porcentaje de Clasificación de Llamadas");
30          console.log("8. Mostrar Cantidad de Llamadas por Calificación");
31          console.log("9. Limpiar datos cargados");
32          console.log("0. Salir");
33
34          console.log("======================================================================");
35
36          this.rl.question("Seleccione una opción: ", (opt) => {
37              this.manejarOpcion(opt.trim());
38          });
39      }
```

- **Opción 1 (Cargar archivo)**
  - Pide al usuario una ruta.
  - Llama a cargarArchivo del CallCenter.
  - Vuelve a mostrar el menú.

```
41      manejarOpcion(opt) {
42          console.log("======================================================================");
43
44          switch (opt) {
45              case "1":
46                  this.rl.question("Ingrese la ruta del archivo que se cargará (ej: Data/ArchivoN.csv): ", (ruta) => {
47
48                      this.callCenter.cargarArchivo(ruta.trim());
49                      this.mostrarMenu();
50                  });
51                  break;
```

- **Opción 2 (Exportar historial)**
  - Llama al método de Reportes para generar el HTML.
  - Vuelve al menú.

Las opciones 3 a 6 son idénticas: llaman a sus respectivos métodos de Reportes.

```
53          case "2":
54              this.reportes.exportarHistorialHTML();
55              this.mostrarMenu();
56              break;
57
58          case "3":
59              this.reportes.exportarOperadoresHTML();
60              this.mostrarMenu();
61              break;
62
63          case "4":
64              this.reportes.exportarClientesHTML();
65              this.mostrarMenu();
66              break;
67
68          case "5":
69              this.reportes.exportarRendimientoHTML();
70              this.mostrarMenu();
71              break;
72
73          case "6":
74              this.reportes.exportAllReports();
75              this.mostrarMenu();
76              break;
```

- **Opción 7 y 8**

Muestra estadísticas en consola (no genera archivo).

```
78          case "7":
79              this.callCenter.mostrarPorcentajeClasificacion();
80              this.mostrarMenu();
81              break;
82
83          case "8":
84              this.callCenter.mostrarCantidadPorEstrellas();
85              this.mostrarMenu();
86              break;
```

- **Opción 9 (Confirmación)**
  - Pide confirmación antes de limpiar.
  - Solo limpia si el usuario responde "s".

```
88          case "9":
89              this.rl.question("¿Seguro que desea limpiar todos los datos cargados? (s/n): ", (resp) => {
90                  if (resp.trim().toLowerCase() === "s") {
91                      this.callCenter.limpiarDatos();
92                      console.log("================================================================");
93                      console.log("Datos limpiados.");
94                  } else {
95                      console.log("================================================================");
96                      console.log("Operación cancelada.");
97                  }
98                  this.mostrarMenu();
99              });
100             break;
```

- **Salida y errores**
  - **Opción 0:** cierra la interfaz y termina.
  - **Default:** si la opción no existe, muestra error y vuelve al menú.
- **Exportación**

Permite que index.js inicie el menú.

```
102             case "0":
103                 console.log("Gracias por su visita, vuelva pronto.");
104                 console.log("================================================================");
105                 this.rl.close();
106                 break;
107
108             default:
109                 console.log("Opción inválida.");
110                 this.mostrarMenu();
111         }
112     }
113 }
114
115 module.exports = Menu;
```

Reportes.js

- **Importaciones**
  - **fs:** para escribir archivos HTML.
  - **path:** para manejar rutas de forma segura.

```
1   const fs = require("fs");
2   const path = require("path");
```

- **Clase Reportes**
  - **Constructor**

Recibe una instancia de CallCenter para acceder a los datos.

```
4   class Reportes {
5       constructor(callCenter) {
6           this.callCenter = callCenter;
7       }
```

- **asegurarCarpetaReportes**

Crea la carpeta Reportes/ si no existe.

- **process.cwd():** ruta actual del proyecto.
- **fs.existsSync y mkdirSync:** verificación y creación.

```
9       asegurarCarpetaReportes() {
10          const dir = path.join(process.cwd(), "Reportes");
11          if (!fs.existsSync(dir)) fs.mkdirSync(dir);
12          return dir;
13      }
```

- **exportarHistorialHTML (inicio)**
  - Obtiene todas las llamadas.
  - Si no hay, muestra mensaje y sale.

```
15      exportarHistorialHTML(filename = "Historial_Llamadas.html") {
16          const llamadas = this.callCenter.getLlamadas();
17          if (llamadas.length === 0) {
18              console.log("No hay llamadas para exportar.");
19              return;
20          }
```

- **Generación de filas**
  - **map():** convierte cada llamada en una fila HTML.
  - **"x".repeat(l.estrellas):** muestra estrellas visuales.
  - **getClasificacion():** agrega la clasificación.

```
22              const rows = llamadas.map(l => `
23      <tr>
24      <td>${l.idOperador}</td>
25      <td>${l.nombreOperador}</td>
26      <td>${l.idCliente}</td>
27      <td>${l.nombreCliente}</td>
28      <td>${"x".repeat(l.estrellas)} (${l.estrellas})</td>
29      <td>${l.getClasificacion()}</td>
30      </tr>`).join("");
```

- **Plantilla HTML**
  - HTML estático con estilo básico.
  - Incluye título, encabezados y tabla.
  - Usa las filas generadas.

```
32        const html = `<!doctype html>
```

```
62    </html>`;
```

- **Escritura del archivo**
  - Guarda el HTML en Reportes/Historial_Llamadas.html.
  - Muestra la ruta en consola.

Los otros métodos (exportarOperadoresHTML, exportarClientesHTML, exportarRendimientoHTML) siguen la misma lógica: generan HTML y lo guardan.

```
64        const dir = this.asegurarCarpetaReportes();
65        const fullPath = path.join(dir, filename);
66        fs.writeFileSync(fullPath, html, "utf8");
67        console.log("Historial exportado a:", fullPath);
68    }
```

- **exportarRendimientoHTML**
  - Ordena operadores por cantidad de llamadas (descendente).
  - Calcula porcentaje: (llamadas / total) * 100.

```
152   exportarRendimientoHTML(filename = "Rendimiento_Operadores.html") {
153       const operadores = Array.from(this.callCenter.getOperadores().values());
154       const total = this.callCenter.getTotalLlamadas();
155       if (operadores.length === 0 || total === 0) {
156           console.log("No hay datos para calcular rendimiento.");
157           return;
158       }
159
160       const operadoresOrdenados = operadores.sort((a, b) => b.cantidadLlamadas() - a.cantidadLlamadas());
```

- **exportAllReports**

Ejecuta los 4 métodos de exportación en secuencia.

- **Exportación**

Permite que Menu.js cree una instancia.

```
210     exportAllReports() {
211         this.exportarHistorialHTML();
212         this.exportarOperadoresHTML();
213         this.exportarClientesHTML();
214         this.exportarRendimientoHTML();
215     }
216 }
217
218 module.exports = Reportes;
219
```

Index.js

Punto de entrada

- Importa la clase Menu.
- Crea una instancia.
- Inicia el sistema.

Este es el archivo que se ejecuta con node index.js.

```
1  const Menu = require("./Servicios/Menu");
2
3  const menu = new Menu();
4  menu.Iniciar();
```

DIAGRAMA DE CLASES

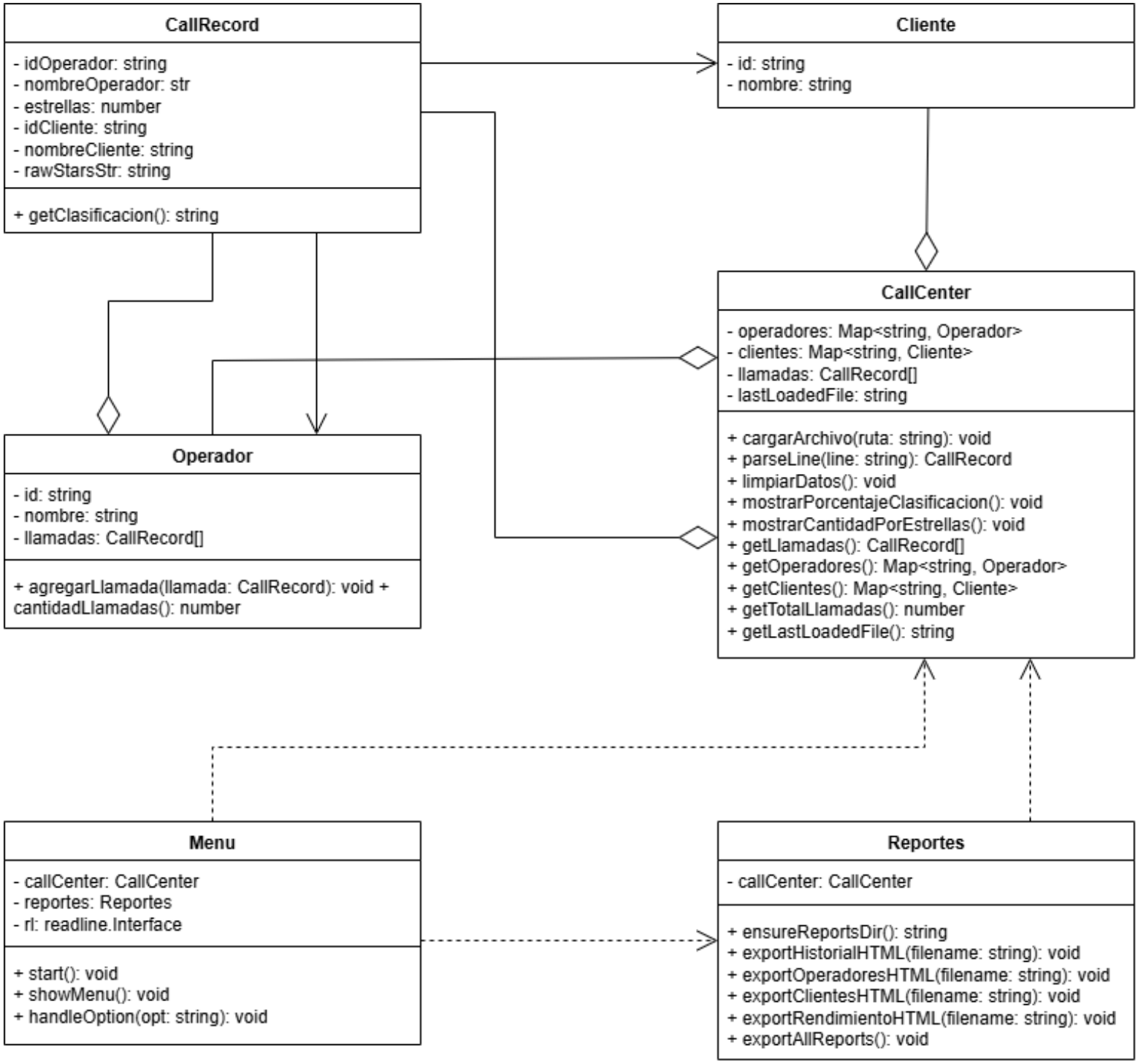

DIAGRAMA DE FLUJO