

Informe de Trabajo Práctico

N°1 - Parte B: Fundamentos - Clases, asociación, dependencia

Programación Orientada a Objetos
Ingeniería en Mecatrónica

Alumno: Juan Manuel BORQUEZ PEREZ
Legajo: 13567



UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO



**FACULTAD
DE INGENIERÍA**

► **1983/2023**
40 AÑOS DE DEMOCRACIA

1 Enunciado

- Prepare un directorio denominado `apellido_legajo_B` (reemplace con sus datos) que contenga 2 subdirectorios denominados `cons_2` y `cons_3`
- Ubique en cada uno de ellos la implementación que corresponda según la consigna.
- Coloque en el directorio raíz (`apellido_legajo_B`) el documento con el informe general para esta entrega (siga el mismo formato de la anterior).
- Al finalizar, suba un archivo comprimido de la carpeta raíz creada (incluyendo todo su contenido) en la actividad denominada Entrega B, dentro del aula virtual.
- En cada consigna, se propone un diagrama de clases mínimo. Siempre que aplique criterios de diseño OO, Usted puede mejorar el mismo, si lo considera conveniente. Si lo hace, agregue al informe, además del diseño ajustado a la implementación realizada, un párrafo descriptivo (dentro del apartado Observaciones) donde justifique sus decisiones.
- Recuerde documentar cada módulo de manera similar al TP anterior. Agregue las guardas.
- Fecha de entrega: 06 de setiembre de 2023.

1.1 Consigna 2

Descargue desde <https://www.bouml.fr/download.html> la OOCASE denominada BOUML e instálela en su computadora.

Replique la secuencia descrita en clase sobre la forma de uso, a los efectos de diseñar el diagrama de clases inicial (que consta de la clase básica de la derecha) y generar de manera automática, los esqueletos de los módulos `.h` y `.cpp`, que le corresponden.

C2File
- filename : String
- N : int
+ serialRequestCSV() : void
+ setFilename(in filename : String) : void
+ setN(in N : int) : void

Incorpore los archivos generados a un proyecto en el IDE C++ de su elección, y luego edítelos, de modo de obtener un programa en lenguaje C++, bajo paradigma orientado a objetos, que resuelva el siguiente requerimiento:

- Lectura de N registros de datos en el puerto serie. Los datos son generados por un controlador basado en Arduino. El sketch a usar para el controlador, se encuentra en el archivo `poo_genera_rand.ino` disponible en el aula virtual.
- Los datos deben solicitarse al controlador en formato CSV. Analice el código provisto para determinar la solicitud a realizar.
- La cantidad de lecturas (N) y el nombre del archivo donde almacenar los datos son dados por el operador.

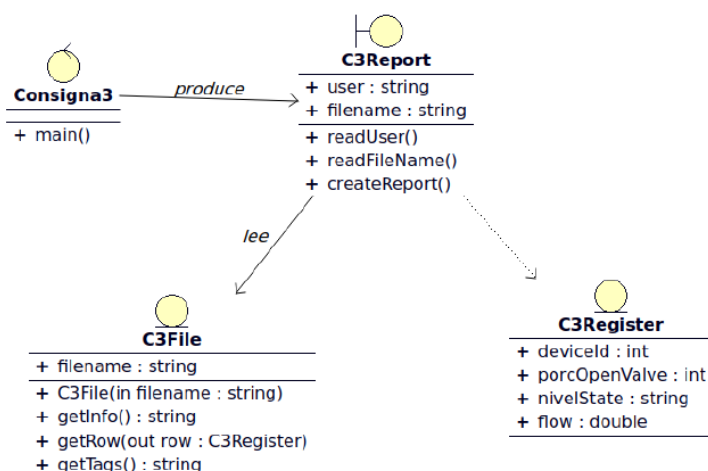
- El archivo a generar es del tipo texto plano, su extensión .csv y su organización interna responde a un formato con línea de encabezado y separado por ; (punto y coma).
- A modo de ejemplo, una supuesta salida para para N=3:

```
dispositivo_id;porcentaje_valvula;estado_nivel;caudal
2;50;"medio";78.15
2;7;"medio";33.90
2;55;"alto";44.42
```

1.2 Consigna 3

Desarrolle un programa en lenguaje Python que, aplicando un paradigma orientado a objetos, resuelva el requerimiento indicado:

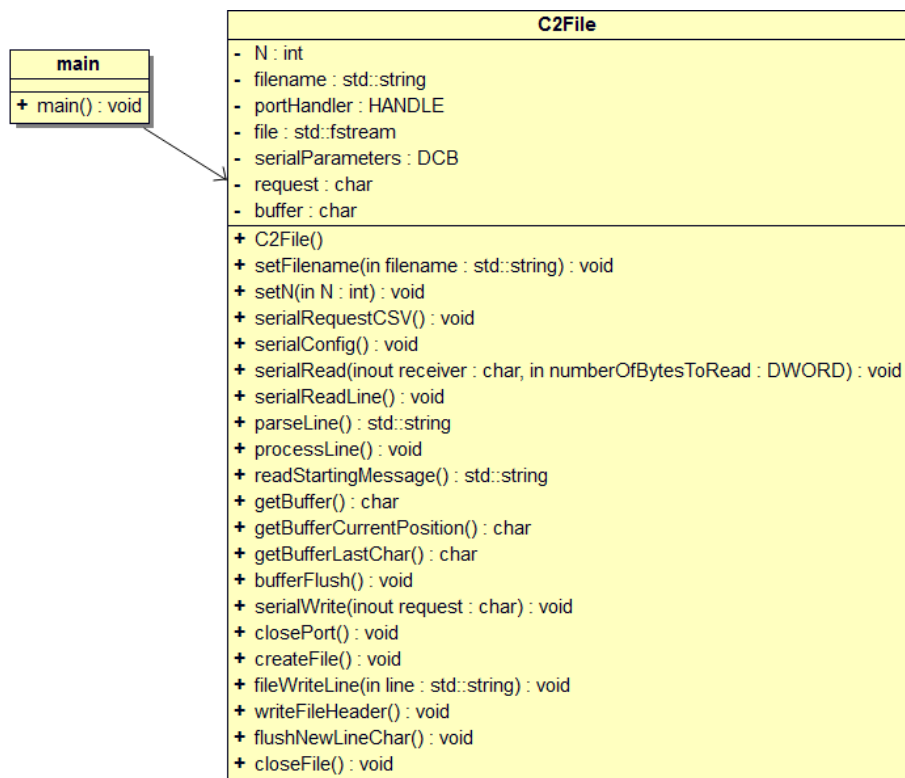
- Lectura de los datos almacenados en un archivo.
- El nombre del archivo es dado por el operador, a partir de un listado de los archivos disponibles en el FS. Prever las situaciones de error que pudieran darse y plantear las acciones que considere adecuadas.
- Despliegue de datos en pantalla, mostrando el nombre del operador actual (quién emite el reporte), la fecha en que los datos fueron almacenados (corresponde tomar la de creación del archivo), el dispositivo desde donde han sido leídos (se muestra sólo una vez) y, finalmente, un listado a 3 columnas para los datos restantes, debidamente alineados. En los encabezados deben mostrarse las etiquetas para las variables de salida, aprovechando las cadenas que se encuentran en el archivo (y que fueran creados por la aplicación de la consigna 2).
- Se usará un diseño de 2 capas con separación entre el modelo y la presentación/control.
- El modelo OO posee las siguientes clases. En caso de considerarlo apropiado, agregue nuevos elementos.



2 Esquema General de la Solución

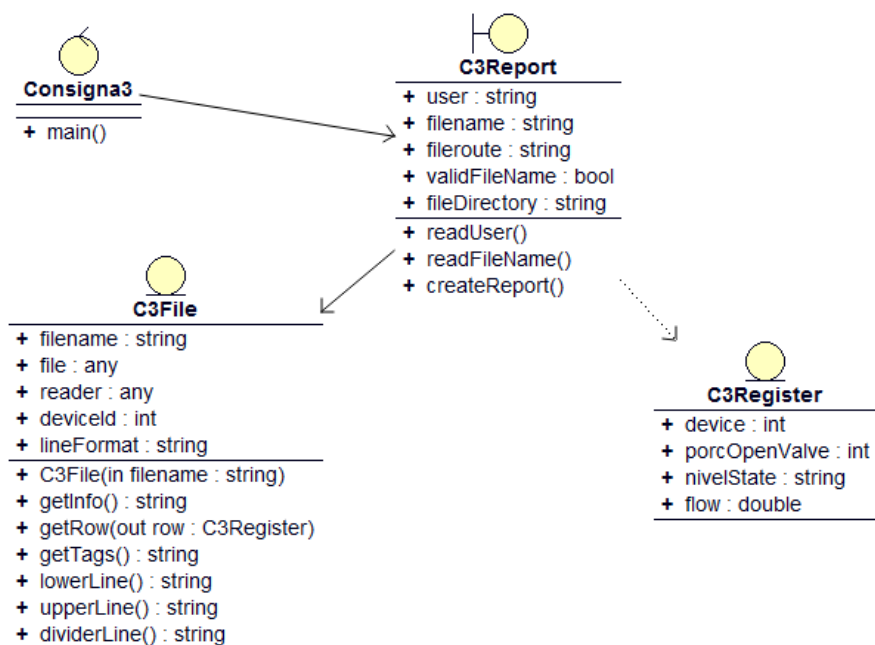
2.1 Consigna 2

Esquema lógico general de la solución para la consigna 2



2.2 Consigna 3

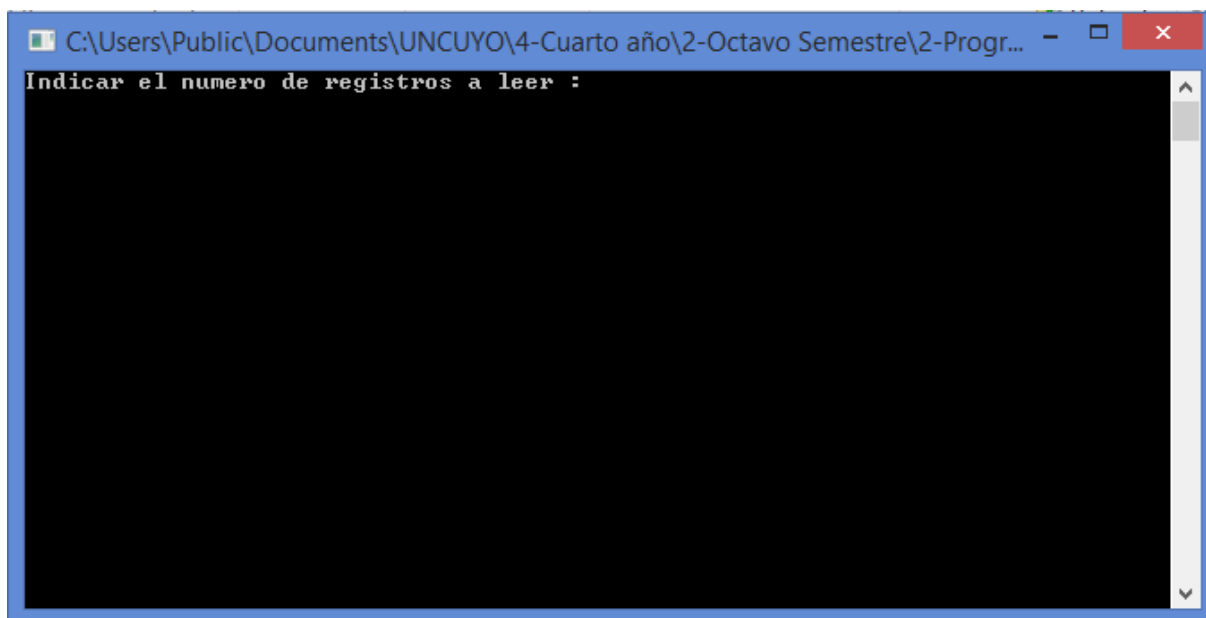
Esquema lógico general de la solución para la consigna 3


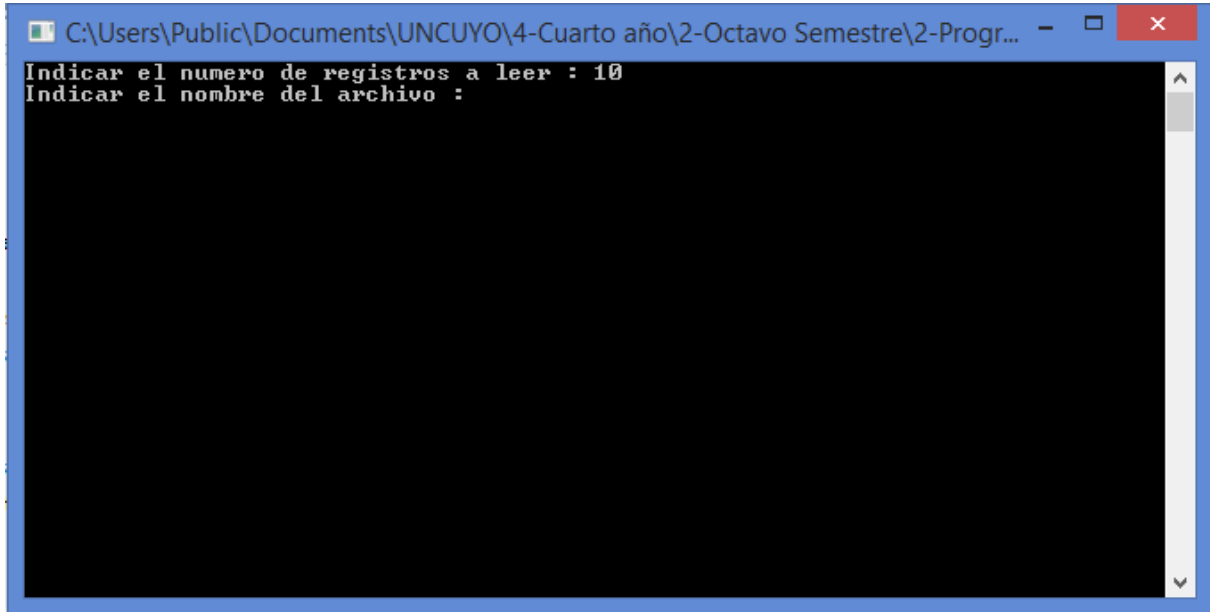


3 Interfaces de Usuario

3.1 Consigna 2

Secuencia de capturas para una ejecución completa





```
C:\Users\Public\Documents\UNCUYO\4-Cuarto año\2-Octavo Semestre\2-Progr... - [X]
Indicar el numero de registros a leer : 10
Indicar el nombre del archivo : dataLogged.csv
Iniciar sketch en la placa arduino...
```

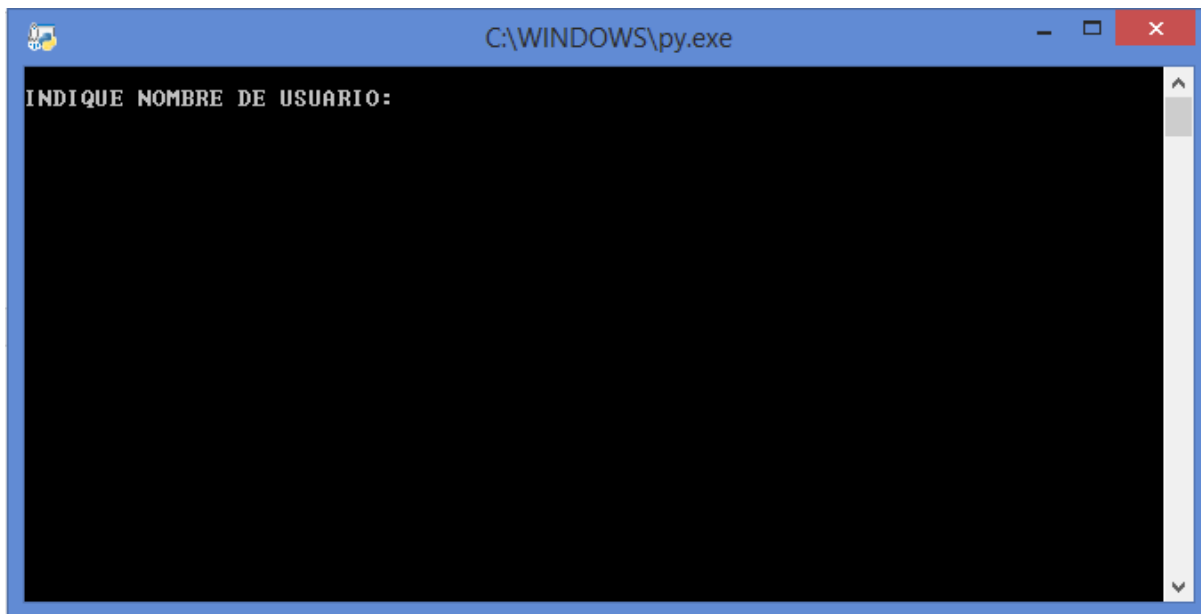
```
C:\Users\Public\Documents\UNCUYO\4-Cuarto año\2-Octavo Semestre\2-Progr... - [X]
Indicar el numero de registros a leer : 10
Indicar el nombre del archivo : dataLogged.csv
Iniciar sketch en la placa arduino...


Inicio de sketch: escucha/genera numero aleatorio
x: cadena XML
j: cadena Json
c: cadena CSU

Lectura...
Finalizado
```


3.2 Consigna 3

Secuencia de Capturas para una ejecución completa





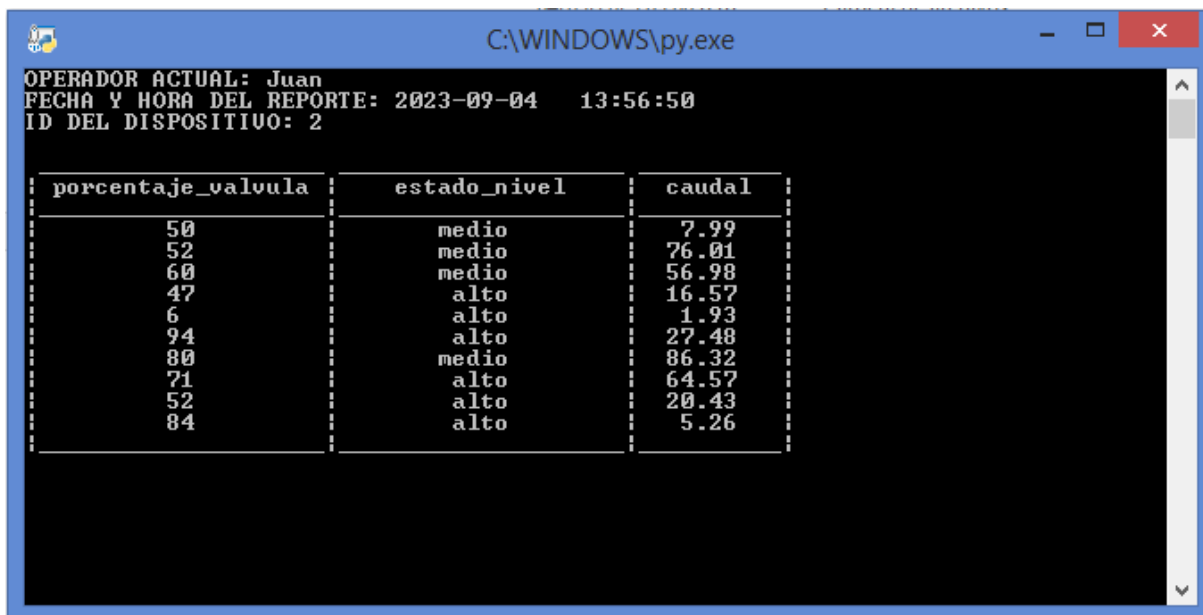
```
C:\WINDOWS\py.exe

INDIQUE NOMBRE DE USUARIO: Juan
DIRECTORIO DEL REPORTE:
.vscode
Arduino_firmware
C2File.cpp
C2File.h
dataLogged.csv
dataLogger.exe
main.cpp
Modelado_UML
INDIQUE NOMBRE DEL ARCHIVO:
```



```
C:\WINDOWS\py.exe

INDIQUE NOMBRE DE USUARIO: Juan
DIRECTORIO DEL REPORTE:
.vscode
Arduino_firmware
C2File.cpp
C2File.h
dataLogged.csv
dataLogger.exe
main.cpp
Modelado_UML
INDIQUE NOMBRE DEL ARCHIVO: dataLogged.csv
```



porcentaje_valvula	estado_nivel	caudal
50	medio	7.99
52	medio	76.01
60	medio	56.98
47	alto	16.57
6	alto	1.93
94	alto	27.48
80	medio	86.32
71	alto	64.57
52	alto	20.43
84	alto	5.26

4 Uso

4.1 Consigna 2

La aplicación permite la obtención de tramas de datos de una placa de desarrollo de Arduino mediante comunicación serial. Los datos se obtienen de la placa mediante tramas de petición enviadas por la aplicación y la placa responde con los datos en un formato específico dependiendo del código enviado en la trama de petición (se utiliza 'c' para formato CSV). La cantidad de tramas de datos a leer es indicada por el usuario. A medida que se reciben las tramas de datos, la aplicación las escribe en formato **CSV** en un archivo de texto plano cuyo nombre es indicado por el usuario y con un encabezado predefinido proporcionado por la aplicación.

La aplicación está **específicamente diseñada para Windows**, ya que hace uso de la **API Win32** para la gestión de la comunicación serial con la placa Arduino. Esto debe tenerse en cuenta en términos de portabilidad si se desea utilizar la aplicación en otro sistema operativo.

Además, la aplicación no realiza un mapeo de los puertos disponibles en la computadora ni identifica automáticamente el puerto en el que se encuentra la placa de Arduino. **Se asume directamente que la placa estará conectada a la computadora en el puerto 'COM 3'**, que es el puerto para el cual se probó la aplicación.

No se ha implementado la validación de los datos de entrada, ya sea en lo que respecta al nombre del archivo indicado para la escritura o al tipo de dato proporcionado para la cantidad de registros a leer. Por lo tanto, el comportamiento en caso de ingresar datos con un tipo incorrecto para cualquiera de estas dos variables es impredecible.

4.2 Consigna 3

La aplicación permite generar un reporte a partir de los datos almacenados en un archivo del sistema de archivos de la computadora, los cuales son generados por un dispositivo

de monitoreo. Estos datos están almacenados en el archivo en formato CSV. Durante la ejecución de la aplicación, se solicita el nombre del operador y el nombre del archivo que contiene los datos. El usuario debe indicar el archivo de una lista con los archivos disponibles en el sistema de archivos.

Los datos para el reporte se presentan en pantalla en forma de tabla y se incluyen metadatos del archivo, como el ID del dispositivo que generó los datos, la fecha de creación del archivo y el nombre del operador proporcionado.

Se han tenido en cuenta situaciones de error que podrían surgir, como la indicación de un nombre de archivo inexistente o incompatible (sin la extensión .csv), así como la posibilidad de que el archivo no contenga datos.

5 Conclusiones

5.1 Cambios Realizados

5.1.1 Consigna 2

La mayoría de los cambios realizados en el código se encuentran indicados en el código fuente de la aplicación como comentarios de línea. Otros de los cambios realizados son los siguientes.

Atributos:

- **portHandler:** Un manejador para el puerto de comunicación serie.
- **file:** Un stream asociado al archivo para la escritura del mismo.
- **Serial Parameters:** Los parámetros dados a la comunicación serie. Se utiliza un bit de Stop, sin bit de paridad, 8 bits de datos y una velocidad de 19200 baudios.
- **request:** Un carácter con el código para la petición de datos en el formato CSV.
- **buffer:** Una cadena de caracteres para almacenar los datos leídos por el puerto serie.

Métodos:

- **serialConfig:** Realiza la configuración de los parámetros de la comunicación serial. Se llama dentro del constructor de la clase.
- **serialRead:** Un método para la lectura por puerto serie de una cantidad indicada de bytes.
- **serialReadLine:** Realiza la lectura de una línea completa por el puerto serie (delimitada por un carácter de fin de línea).
- **parseLine:** Da formato CSV a las líneas leídas por puerto serie.
- **processLine:** Realiza la lectura, parseo y escritura en archivo de las líneas leídas.
- **readStartingMessage:** Realiza la lectura del mensaje de inicio en el sketch de la placa Arduino.

- **getBuffer:** Obtiene el contenido del buffer.
- **getBufferCurrentPosition:** Obtiene un punto a la posición del primer carácter nulo en el buffer.
- **getBufferLastChar:** Obtiene el último carácter no nulo en el buffer.
- **bufferFlush:** Coloca caracteres nulos en todas las posiciones del buffer.
- **serialWrite:** Escribe una cadena de caracteres por el puerto serie.
- **closePort:** Cierra la comunicación con el puerto serie.
- **createFile:** Crea el archivo para almacenar los datos.
- **fileWriteLine:** Escribe una línea en el archivo.
- **writeFileHeader:** Escribe el encabezado del archivo CSV.
- **flushNewLineChar:** Descarta los caracteres de nueva línea que envía la placa con cada petición.
- **closeFile:** Cierra el archivo.

Como se puede observar se prefirió separar las operaciones en varios métodos de la clase en lugar de llevar a cabo todas las operaciones dentro de un mismo método **serialRequestCSV** para dar más legibilidad al código y hacer más sencilla la programación.

5.1.2 Consigna 3

En **C3Report:**

Atributos:

- **fileroute:** La ruta de la carpeta que contiene el archivo **.csv**.
- **fileDirectory:** Una lista de cadenas de caracteres con los nombres de todos los archivos en el directorio del archivo.
- **validFileName:** Una bandera para determinar si el nombre del archivo ingresado por el operador es válido o no.

En **C3File:**

Atributos:

- **file:** El archivo en sí o un manejador del archivo.
- **reader:** Un objeto para recorrer el archivo obtenido de la librería "csv".
- **deviceId:** Una variable que indica el ID del dispositivo que generó los datos del archivo.
- **lineFormat:** Una cadena de caracteres para aplicar formato a las filas de la tabla.

Métodos:

- Los últimos tres métodos que se indican en el diagrama lógico para la consigna 3 se utilizan para obtener las líneas de borde superior, inferior y las que separan los campos de la tabla de las filas con datos.

5.2 Dificultades Encontradas

Se tuvieron **inconvenientes en la lectura de los datos** por el puerto serie debido a que en la lectura con la función `ReadFile` proporcionada por la API se requiere especificar la cantidad de bytes a leer que se almacenarán en el buffer. No obstante, dado que no se conoce de antemano el tamaño exacto de las líneas de datos enviadas por la placa, no es posible proporcionar esta indicación a priori.

Si se intenta especificar una cantidad de bytes mayor que la longitud máxima de una línea enviada por la placa teniendo en cuenta lo dicho, la función se bloqueará durante un tiempo (aproximadamente 5 segundos), como está predefinido en los `TIMEOUTS` de la comunicación, esperando la cantidad de bytes especificada. Esto ralentizará significativamente la solicitud de datos.

Para abordar este problema, en la lectura de una trama de datos enviada por la placa, se leen los caracteres de uno en uno hasta encontrar el primer carácter de fin de línea. De esta manera, la función `ReadFile` espera la lectura de un solo carácter en lugar de una cantidad de bytes fija, lo que evita el bloqueo prolongado y mejora la velocidad de la solicitud de datos.

5.3 Comentario/Extensiones

En la **Consigna 2**, sería posible llevar a cabo la toma de datos de sensores reales en lugar de generar los datos de forma aleatoria, aunque esto no sería útil para demostrar o abordar los contenidos específicos de la materia.

Se podría realizar el mapeo de los puertos de comunicación serie disponibles en el ordenador en busca de aquel en el que se encuentra conectado el dispositivo que genera los datos.

Además, podría investigarse la posibilidad de hacer que la aplicación sea compatible con otros sistemas operativos, reduciendo así la dependencia de la API de Windows. Para lograr esto, inicialmente, sería necesario investigar acerca de las bibliotecas más utilizadas para el manejo de la comunicación serie en C++.

También se podría haber logrado una mayor separación de responsabilidades mediante la incorporación de más clases en la aplicación.

Finalmente, para ambas consignas, se podría haber logrado un mejor manejo de las excepciones que ocurren en el programa.

5.4 Valoración Personal/Observaciones

Considero que he logrado cumplir con los requisitos establecidos en las consignas aplicando los conceptos de Programación Orientada a Objetos. Entre los aspectos que destacaría:

- Es necesario realizar una investigación más exhaustiva sobre las bibliotecas utilizadas para la comunicación serial en C++, especialmente aquellas que son ampliamente adoptadas por la comunidad. Esto permitiría reducir la dependencia de una API específica de un sistema operativo y brindaría mayor portabilidad a la aplicación.
- En ambas soluciones, se podría mejorar la gestión de las excepciones que puedan surgir durante la ejecución del programa.
- Se podría lograr una mayor separación de responsabilidades mediante la incorporación de más clases en el diseño.
- Es importante profundizar en la comprensión de las relaciones entre clases a través de la práctica.
- Se requiere un mejor entendimiento de los estereotipos de clases, como control, interfaz y entidad, así como su implementación.

[7], [12], [9], [11], [4], [5], [2], [3], [6], [8], [10], [14], [13], [1]

Referencias

- [1] Ratón de Biblioteca. *El Bibliotecario: Enciclopedia de C y C++*. URL: <https://www.ratondbiblioteca.com/ebc>.
- [2] Microsoft Docs. *COMMTIMEOUTS structure*. URL: <https://learn.microsoft.com/en-us/windows/win32/api/winbase/ns-winbase-commtimeouts>.
- [3] Microsoft Docs. *GetCommTimeouts function*. URL: <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-getcommtimeouts>.
- [4] Microsoft Docs. *Opening a File for Reading or Writing*. URL: <https://learn.microsoft.com/en-us/windows/win32/fileio/opening-a-file-for-reading-or-writing>.
- [5] Microsoft Docs. *ReadFile function*. URL: <https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-readfile>.
- [6] Microsoft Docs. *SetCommTimeouts function*. URL: <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-setcommtimeouts>.
- [7] Python Official Documentation. *CSV File Reading and Writing*. URL: <https://docs.python.org/es/3/library/csv.html>.
- [8] GeeksforGeeks. *File Handling in C++*. URL: <https://www.geeksforgeeks.org/file-handling-c-classes/>.
- [9] Miro. *¿Qué es un diagrama UML?* URL: <https://miro.com/es/diagrama/ques-diagrama-uml/>.
- [10] MyGreatLearning. *File Handling in C++*. URL: <https://www.mygreatlearning.com/blog/file-handling-in-cpp/>.

- [11] Universidad del País Vasco. *Introducción a UML*. URL: <http://www.vc.ehu.es/jiwotvim/IngenieriaSoftware/Teoria/BloqueII/UML-3.pdf>.
- [12] Real Python. *Reading and Writing CSV Files in Python*. URL: <https://realpython.com/python-csv/>.
- [13] YouTube - RishabhPandey. *File Handling in C++*. URL: https://www.youtube.com/watch?v=x_5g8-i7rlw&ab_channel=RishabhPandey.
- [14] YouTube - rmb1905. *File Handling in C++*. URL: https://www.youtube.com/watch?v=0hAqfaaZpes&ab_channel=rmb1905.