TRABAJO PRACTICO Nº 2 – Desarrollo Orientado a Objetos

Objetivo general:

Implementar programas orientados a objetos de complejidad básica/media en lenguaje C++.

Consigna 1

Motivación:

Esta actividad tiene la finalidad de definir una clase (al menos) que permita resolver un requerimiento sencillo e implementarla en lenguaje C++.

La definición de las entidades que conforman un sistema no es rígida. Si bien existen algunos aspectos comunes o estándares, existen otros que son propios del entorno donde existe esa entidad. Por ejemplo, una entidad denominada Usuario, que representa a un actor de un sistema bancario puede tener características diferentes a las de ese actor en un sistema académico.

Una clase se define en base a un conjunto de características, diferenciadas en atributos/propiedades/campos y en comportamientos/operaciones/métodos/funciones.

Las clases surgen, fundamentalmente, como abstracción del desarrollador, al detectar los objetos reales que existen en un sistema, o bien de conceptualizaciones de diseño.

Las clases sirven no sólo como concepto que identifica y agrupa a un conjunto de objetos similares presentes en un sistema, sino también como plantilla o matriz desde la cual obtener objetos nuevos de ese tipo.

Los atributos se implementan a través de variables, capaces de contener datos que describen el estado o la información que maneja la entidad.

Los comportamientos se implementan a través de métodos/funciones, capaces de gestionar los datos propios de la entidad.

Los objetos (virtuales) aparecen en el sistema, cuando el programa construido por el desarrollador, se ejecuta en un equipo de cómputo.

Para que los objetos surjan, es necesario que el desarrollador codifique instrucciones que instancien los mismos, usando para ello las definiciones de clases realizada previamente.

¿Qué criterios seguir para decidir qué atributos corresponden a una clase?

Hágase una o varias de estas preguntas guía:

- ¿este dato describe al objeto en el mundo real o en el contexto de mi aplicación?
- ¿este dato del problema representa una característica esencial/propia del objeto?
- ¿este dato pertenece naturalmente al objeto, o está relacionado con otra entidad?
- si coloco este atributo ¿encaja o estoy mezclando con responsabilidades de otra entidad del sistema?
- ¿qué necesita saber este objeto para cumplir su función?
- ¿qué información describe su estado actual?
- ¿qué datos estarán disponibles al momento de crear el objeto?

¿Qué criterios seguir para decidir qué comportamientos corresponden a una clase?

Hágase una o varias de estas preguntas guía, y agregue la operación si la respuesta es SÍ:

- ¿esta operación depende de los datos internos del objeto?
- ¿tiene sentido que este objeto realice esta acción?
- ¿esta operación pertenece a la responsabilidad de la clase?
- ¿esta operación modifica o consulta el estado del objeto?

Requerimientos:

Desarrollar una herramienta software capaz de crear un archivo de texto o mostrar el contenido del mismo, con datos que han sido generados por un dispositivo Arduino (o compatible).

Los datos se almacenan en el archivo, organizados conforme a alguno de los estándares CSV.

La presentación de los datos en pantalla debe hacerse en formato CSV o JSON o XML, según opciones provistas por el operador.

La salida debe mostrar, primero los datos propios del archivo en forma de tabla horizontal (etiqueta → dato) y luego los datos contenidos en forma de tabla vertical (grupo de filas con encabezado único y datos encolumnados).

Tanto el nombre del archivo, la selección del modo de trabajo de lectura (recupera datos de un archivo existente) o escritura (crea un archivo a partir los datos recibidos desde el generador Arduino), como los parámetros auxiliares (por ejemplo, para el caso de modo escritura: tipo de formato [x / j / c] de recepción esperado, cantidad de lecturas a realizar) son provistos por línea de comandos.

Para ello, usar el formato de opciones de trabajo habituales en interacciones mediante CLI al estilo Linux.

Los archivos deben crearse/almacenarse en un directorio específico y la extensión se fija por código.

La implementación Orientada a Objetos realizada debe separar la capa de presentación/control de la de modelo.

Es conveniente que la implementación utilice excepciones para gestionar los errores y/o devolver mensajes destinados al usuario o a su almacenamiento.

Procedimiento general:

1) Descargar el firmware y colocarlo en un dispositivo Arduino.

2) Analizar el código del mismo y ejecutarlo, tanto para comprender su comportamiento como para determinar los datos producidos. Puede usar para ello la consola serie.

3) Diseñar en Umbrello un modelo OO usando UML, que muestre y defina la/s clase/s fundamentales (considere el anexo provisto y los lineamientos dados en TP 1 - Actividad 4 - item 7).

4) Generar los módulos correspondientes, con el código base en lenguaje C++

5) Editar y completar la implementación de modo de obtener una aplicación funcional que resuelva el requerimiento.

6) Realizar un informe similar al documento provisto: prav_estructuraTP2.pdf

7) Preparar una carpeta en el FS con archivos depurados, similar al ejemplo explicado en clase, que permita luego la compresión para su entrega.

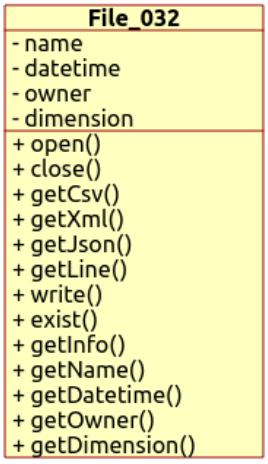
Recursos complementarios (disponibles en el aula virtual)**:**

- Firmware productor de datos para dispositivo Arduino (o compatible).
- Informe de ejemplo.
- Documentos de clase: Interfaz de línea de comandos (poo_cli_v24.pdf), Gestion de excepciones (poo_excepciones_basicov2404.pdf) y manejo de archivos (prav_25_serializacion.pdf)
- Códigos de ejemplo: encolumnado de datos en C++
- Diagrama de clases en anexo (mínimo sugerido considerando posibles extensiones posteriores).

Anexo a consigna 1

Diagrama mínimo sugerido

- Se muestra sólo la clase fundamental, con atributos y operaciones sin firmas.
- No se han incluido constructores
- Otros detalles quedan a buen criterio y libertad del desarrollador.
- Observe el agregado de los últimos 3 dígitos del legajo al nombre de la clase.



Descripción de Atributos de la clase Archivo (File_NNN):

- name: El nombre del archivo (puede incluir o no su extensión).
- datetime: La fecha/hora en que el archivo fue creado.
- owner: Es el usuario creador del archivo.
- dimension: Cantidad de líneas (de texto) almacenadas.

Descripción de Métodos de la clase Archivo (File_NNN):

- open(): abre el archivo para lectura/escritura. Puede tomar un parámetro para definir el modo de apertura (por ejemplo, "r" para lectura, "w" para escritura).
- close(): cierra el archivo, liberando recursos del sistema.
- getCsv(): devuelve el contenido completo (ya almacenado como CSV) del archivo luego de la lectura. Podría tener argumentos de rango.
- getJson(): devuelve el contenido completo en formato Json. Podría tener argumentos de rango.
- getXml(): devuelve el contenido completo en formato XML. Podría tener argumentos de rango.
- getLine(): devuelve el contenido de una línea leida del archivo. Podría tomar como parámetro la línea que se desea recuperar. Sin argumentos podría leer la siguiente línea disponible.
- write(): escribe datos al archivo. Podría tomar como parámetro los datos que se quieren escribir.
- exist(): Verifica si el archivo existe en la ruta especificada.
- getNombre(): Devuelve el nombre del archivo.
- getFecha(): Devuelve la fecha de creación del archivo.
- getpropietario(): Devuelve el nombre del archivo.
- getDimension(): Devuelve la cantidad de líneasdel archivo.
- getInfo(): Devuelve información relativa al archivo (es el conjunto de los 4 anteriores incluyendo cada etiqueta).
- Constructor completo
- Constructor vacío
- Constructor sólo con nombre y fecha

Preguntas que ayudan al momento de optimizar definiciones:

- ¿puedo evitar este atributo por ser derivado (se puede calcular a partir de otros) o redundante?
- ¿este dato se repite en muchos objetos similares? (posible atributo compartido o clase separada)
- ¿este dato tiene cierta permanencia en el ciclo de vida del objeto o cambia con gran frecuencia?
- ¿esta operación involucra sólo a esta clase, o también a otras?