Reporte Programación avanzada

Integrantes: Maximiliano Bustamante
Sebastían Cruz
Joaquín fuenzalida

Tema 5: Gestión de Asistencia de alumnos de un colegio

El proyecto es una aplicación Java que gestiona la información de cursos escolares a través de la manipulación de archivos CSV. Facilita la administración de alumnos y el seguimiento de asistencias, ofreciendo funcionalidades para:

Cargar y actualizar datos de alumnos: Permite leer y escribir la información de alumnos como nombre, apellido y RUT desde archivos CSV.

Gestionar registros de asistencia: Actualiza y consulta la asistencia de los estudiantes para diferentes fechas, utilizando archivos CSV específicos para almacenar esta información.

Calcular asistencias promedio: Proporciona herramientas para calcular el promedio histórico de asistencia y listar cursos con asistencia superior a un umbral definido.

La aplicación utiliza excepciones personalizadas para manejar errores específicos relacionados con la conversión de datos y el acceso a archivos, asegurando un funcionamiento robusto y confiable.

● SIA1.2 Diseño conceptual de clases del Dominio y su código en Java

- **Clase Alumno**
  Alumno.java
  Descripción: Representa a un alumno con atributos básicos como RUT, nombre y apellido.

- **Clase Curso**
  Curso.java
  Descripción: Gestiona un curso, conteniendo información como el nombre del curso, una lista de alumnos (List), y un registro de asistencias por fecha.

- **Clase GestorCSV**
  GestorCSV.java
  Descripción: Proporciona funcionalidades para interactuar con archivos CSV, como cargar fechas, cargar alumnos, añadir y eliminar alumnos.

- **Clase RegistroAsistencia**

  RegistroAsistencia.java

  Descripción: Maneja el registro de asistencia de los alumnos, posiblemente conteniendo la fecha, una lista o conteo de alumnos presentes, etc

- **Clase CursoOpciones**

  CursoOpciones.java

  Descripción: Parece ser una clase que ofrece opciones de configuración o manejo específico para cursos, aunque los detalles exactos dependen de su implementación interna.

- **Clase MainFrame**

  Archivo: MainFrame.java

  Descripción: Proporciona la interfaz gráfica principal del usuario, gestionando diferentes paneles y opciones del programa.

- **Clase SpinnerDatePicker**

  Archivo: SpinnerDatePicker.java

  Descripción: Implementa un selector de fecha personalizado, utilizando en la interfaz gráfica para seleccionar fechas.

- **Clase ConexionFallidaException**

  Archivo: ConexionFallidaException.java

  Descripción: Define una excepción personalizada que se lanza cuando falla una conexión, posiblemente en contextos de bases de datos o redes.

- **Clase ValorInvalidoException**

  Archivo: ValorInvalidoException.java

  Descripción: Define una excepción personalizada para manejar errores relacionados con valores inválidos en la entrada de datos.

- **Clase PROYECTOSIA1**

  Archivo: PROYECTOSIA1.java

  Descripción: Clase main que ejecuta el programa.

● SIA1.3 Todos los atributos de todas las clases deben ser privados y poseer sus respectivos métodos de lectura y escritura (getter y setter).

Clase Alumno

- **Atributos privados:** rut, nombre, apellido
- **Implementación de getters y setters:**
  getRut(), setRut(String rut)
  getNombre(), setNombre(String nombre)
  getApellido(), setApellido(String apellido)

Estos métodos aseguran una correcta encapsulación de los datos del alumno, permitiendo acceso y modificación controlados de sus atributos.

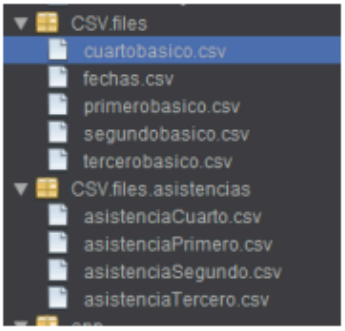
Clase Curso

- **Atributos privados: nombre, totalAlumnos**
- **Implementación de getters y setters:**
  getNombre(), setNombre(String nombre)
  getTotalAlumnos(), setTotalAlumnos(int totalAlumnos)

Estos métodos aseguran una correcta encapsulación de los datos del alumno, permitiendo acceso y modificación controlados de sus atributos

- **SIA1.4 Se deben incluir datos iniciales dentro del código.**

Se crearon 3 tipos de csv donde incluimos datos iniciales los cuales son conformados de la siguiente manera:



CSV fecha: Este csv se utiliza principalmente como calendario general que tiene la función de hacernos saber en que dia estamos de forma simulada y la cual su formado seria dd/mm/aa.

```
31/03/24
01/04/24
02/04/24
03/04/24
04/04/24
```

CSV cursos: Existe 4 csv de este tipo los cuales son de primero, segundo, tercero y cuarto básico los cuales contiene los datos de cada alumno y su formato es rut/nombre/apellido.

```
22098765-3;Carlos;Salazar
22345678-4;Sofia;Perez
22567890-5;Luis;Gonzalez
22678901-6;Ana;Rodriguez
22456789-7;Diego;Hernandez
22234567-8;Andrea;Fernandez
22901234-9;Pablo;Martinez
22789012-1;Laura;Lopez
22345678-2;Alejandro;Gomez
```

CSV asistencia: Existe 4 csv de este tipo los cuales son de asistencia primero , asistencia segundo, asistencia tercero y asistencia cuarto básico, el cual tendrá la fecha al cuales estará sincronizada con el **csv fechas** y cumplirá el rol de guardar la cantidad de alumnos que fueron el presente día. Y su formato es dd/mm/aa/asistencia.

```
31/03/24
01/04/24;21
02/04/24
03/04/24
04/04/24
05/04/24
```

- **SIA1.5 Diseño conceptual y codificación de 2 colecciones de objetos, con la 2ª colección anidada como muestra la figura. Las colecciones pueden ser implementadas mediante arreglos o clases del Java Collections Framework (JCF).**

Primera Colección: Lista de Alumnos:

En el método **cargarAlumnosDesdeCSV(String rutaArchivo)**, utilizas una **List<Alumno>** para almacenar los datos de los alumnos cargados desde un archivo CSV. Esta lista representa la primera colección en tu diseño.

```
public List<Alumno> cargarAlumnosDesdeCSV(String rutaArchivo) {
    List<Alumno> alumnos = new ArrayList<>();

    // Código para cargar datos desde el archivo CSV y añadirlos a la lista

    return alumnos;
}
```

Ubicación en el código: Dentro del método **cargarAlumnosDesdeCSV** en **GestorCSV.java.**

Segunda Colección: Registros de Asistencia

La segunda colección, que es anidada, está conceptualizada en la clase **Curso**. La colección anidada estaría en el manejo de los registros de asistencia, donde asocias una **fecha (LocalDate**) con un registro de asistencia para esa **fecha (RegistroAsistencia)**, probablemente a través de un **Map<LocalDate, RegistroAsistencia>.**

```
public class Curso {

    private Map<LocalDate, RegistroAsistencia> asistenciasPorFecha;
```

Ubicación en el código: Esta implementación estaría en la clase **Curso,** específicamente en el atributo **asistenciasPorFecha.**

- **SIA1.6 Diseño conceptual y codificación de 2 clases que utilicen sobrecarga de métodos (no de constructores)**

1. La clase GestorCSV utiliza la sobrecarga de métodos para proporcionar diferentes maneras de cargar y manipular datos de alumnos desde archivos CSV.

- **Sobrecarga Implementada:**

- **Método para cargar todos los alumnos:** public List<Alumno> cargarAlumnosDesdeCSV(String rutaArchivo)
- **Método sobrecargado para cargar un alumno específico por RUT:** public Alumno cargarAlumnoPorRUT(String rutaArchivo, String rutBuscado)
- **Método sobrecargado para cargar alumnos por nombre:** public List<Alumno> cargarAlumnosPorNombre(String rutaArchivo, String nombreBuscado)

- Estos métodos permiten cargar información de alumnos de manera general o más específica según el RUT o nombre, demostrando la flexibilidad que ofrece la sobrecarga de métodos.

2.La clase GestorCSV

Sobrecarga Implementada:

Cargar todos los alumnos: public List<Alumno> cargarAlumnosDesdeCSV(String rutaArchivo)

Carga una lista completa de alumnos desde un archivo CSV.
Cargar un alumno por RUT: public Alumno cargarAlumnoPorRUT(String rutaArchivo, String rutBuscado)

Busca y carga la información de un alumno específico utilizando su RUT.
Cargar un alumno por Nombre y Apellido: public Alumno cargarAlumnoPorNombreApellido(String rutaArchivo, String nombreBuscado, String apellidoBuscado)

Permite buscar y cargar la información de un alumno basándose en su nombre y apellido.

- **SIA1.7 Diseño conceptual y codificación de al menos 1 clase mapa del Java Collections Framework**

Clase curso

Uso de Map: El atributo **"asistenciasPorFecha"** en la clase "**Curso"** refleja directamente el uso de un mapa del JCF.

Clave-Valor: El mapa utiliza LocalDate como clave, lo que permite una organización eficiente y accesible de los registros por fecha. El valor asociado a cada clave es una instancia de **"RegistroAsistencia"**, que puede contener detalles como los alumnos presentes o ausentes ese día.

```
    private int totalAlumnos;
    private Map<LocalDate, RegistroAsistencia> asistenciasPorFecha;
```

- **SIA1.8 Se debe hacer un menú para el Sistema donde ofrezca las funcionalidades de: 1) Inserción Manual / agregar elemento y 2) Mostrar por pantalla listado de elementos. Esto para la 2ª colección de objetos (colección anidada) del SIA1.5**

Al iniciar el código vemos en un principió el día actual con su fecha al lado y nos pide elegir un curso para gestionar o cambiar de día para cambiar datos de ser necesarios.

```
run:
Día: domingo 31/03/24

Elija un curso:
1. Primero Basico
2. Segundo Basico
3. Tercero Basico
4. Cuarto Basico
8. Dia anterior
9. Dia siguiente
0. Salir
Ingrese una opción:
```

- Luego de **elegir un curso** (Primero Básico en este caso) vemos el siguiente menú de gestión.

```
Menu Curso - Primero Básico:
1. Marcar asistencia
2. Ver alumnos
3. Ver asistencia histórico
4. Agregar alumno
5. Eliminar alumno
0. Volver al menú principal
Ingrese una opción:
```

- **Al agregar un alumno** nos pedirá su rut, nombre y apellido:

```
Ingrese una opción: 4
Ingresa el RUT del alumno: 21031596-9
Ingresa el nombre del alumno: Maximiliano
Ingresa el apellido del alumno: Bustamante
Alumno agregado exitosamente.
```

- Al **ver alumnos** se nos mostrará una lista de los alumnos con los ruts, nombres y apellidos del curso:

```
Alumnos en Primero Básico:
0. 22123456-2 - Maria Gonzalez
1. 22098765-3 - Carlos Rodr□guez
2. 22345678-4 - Sof□a Hern□ndez
3. 22567890-5 - Luis Fern□ndez
4. 22678901-6 - Ana Mart□nez
5. 22456789-7 - Diego L□pez
6. 22234567-8 - Andrea G□mez
7. 22901234-9 - Pablo D□az
8. 22789012-1 - Laura Ruiz
9. 22345678-2 - Alejandro S□nchez
10. 22567890-3 - Carmen Ortiz
11. 22123456-4 - Javier Rojas
```

- **SIA1.9 Todas las funcionalidades deben ser implementadas mediante consola (Sin ventanas)**

Luego de **elegir un curso** (Primero Básico en este caso) vemos el siguiente **menú de gestión.**

```
Menu Curso - Primero Básico:
1. Marcar asistencia
2. Ver alumnos
3. Ver asistencia histórico
4. Agregar alumno
5. Eliminar alumno
0. Volver al menú principal
Ingrese una opción:
```

- **Al agregar un alumno nos pedirá su rut, nombre y apellido:**

```
Ingrese una opción: 4
Ingresa el RUT del alumno: 21031596-9
Ingresa el nombre del alumno: Maximiliano
Ingresa el apellido del alumno: Bustamante
Alumno agregado exitosamente.
```

- **Al eliminar un alumno nos preguntará si queremos buscar el alumno a través de su nombre y apellido o rut:**

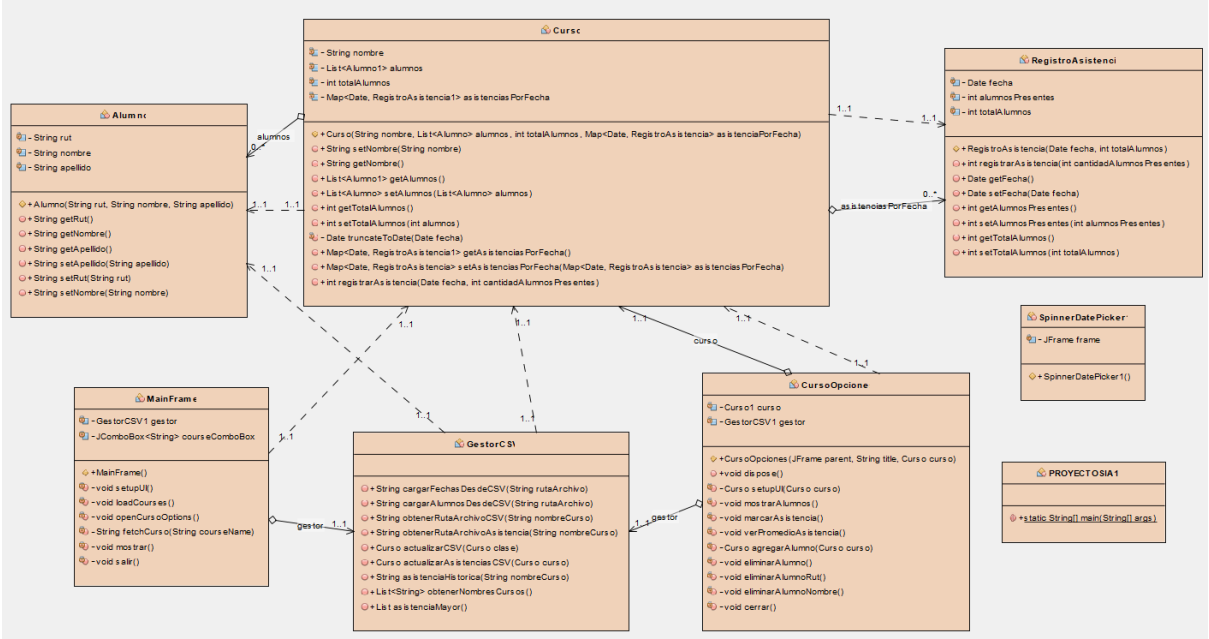
```
Ingrese una opción: 5
¿Deseas eliminar por RUT o por nombre y apellido? (R/N)
R
Ingresa el RUT del alumno a eliminar:
21031596-9
Alumno con identificador '21031596-9' eliminado exitosamente.
```

- **Al marcar asistencia el código nos pedirá ingresar la cantidad de alumnos presentes en el día:**

```
Ingrese una opción: 1
Ingrese la cantidad de alumnos presentes:
21
```

SIA2.1 Diseño de diagrama de clases UML. El diagrama debe contener las clases propias y utilizadas en el proyecto, indicando los elementos propios de cada clase (Atributos, métodos, etc.) y asociaciones entre clases indicando su cardinalidad cuando corresponda. El diagrama debe presentarse como una o más imágenes legibles dentro del reporte. Cada clase debe aparecer al menos en 1 de las imágenes en su forma expandida (no colapsada).

Diagrama Completo:

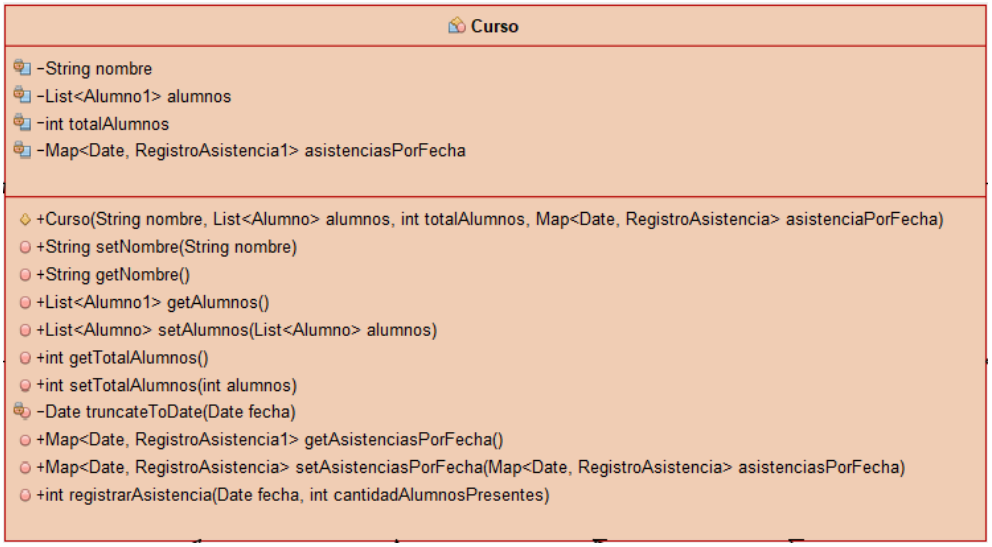


El diagrama UML contiene las clases propias utilizadas en el proyecto se indican los elementos propios de estas y sus cardinalidades entre sí.

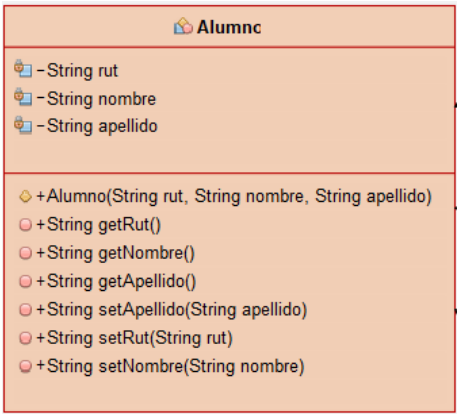
En el caso de la clase **`SpinnerDatePicker`** ésta no se conecta con el resto del diagrama debido a que es esencialmente una clase independiente diseñada para ofrecer una funcionalidad específica y con la clase **`PROYECTOSIA1`** no está conectada al resto del diagrama UML porque actúa como el punto de entrada de la aplicación, conteniendo el método main que inicializa y configura el sistema, sin establecer relaciones directas de dependencia o asociación con otras clases.

Clases:

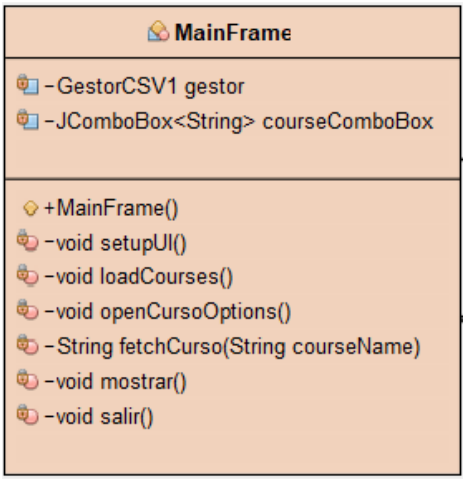
- **Curso:**



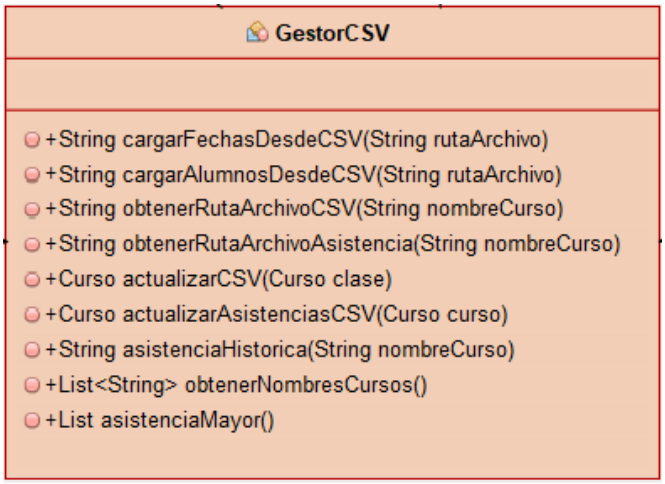
- **Alumno**



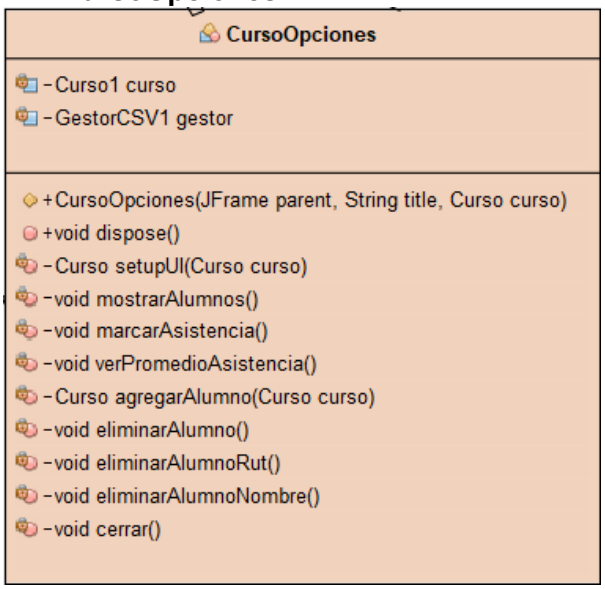
- **Mainframe**



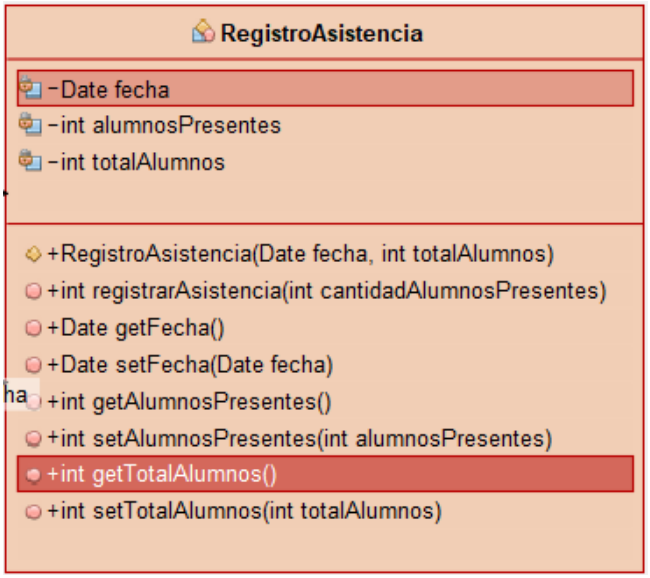
- **GestorCsv**



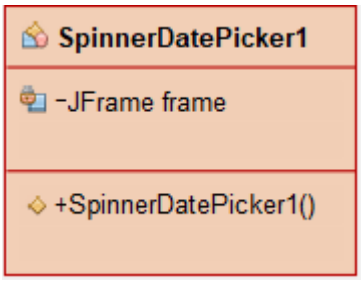
- **CursoOpciones**



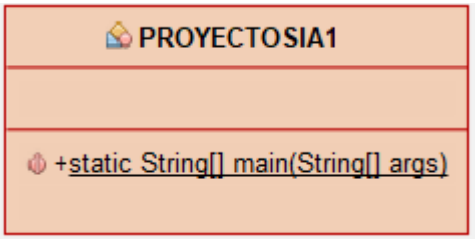
- **RegistroAsistencia**



- **SpinnerDatePicker**



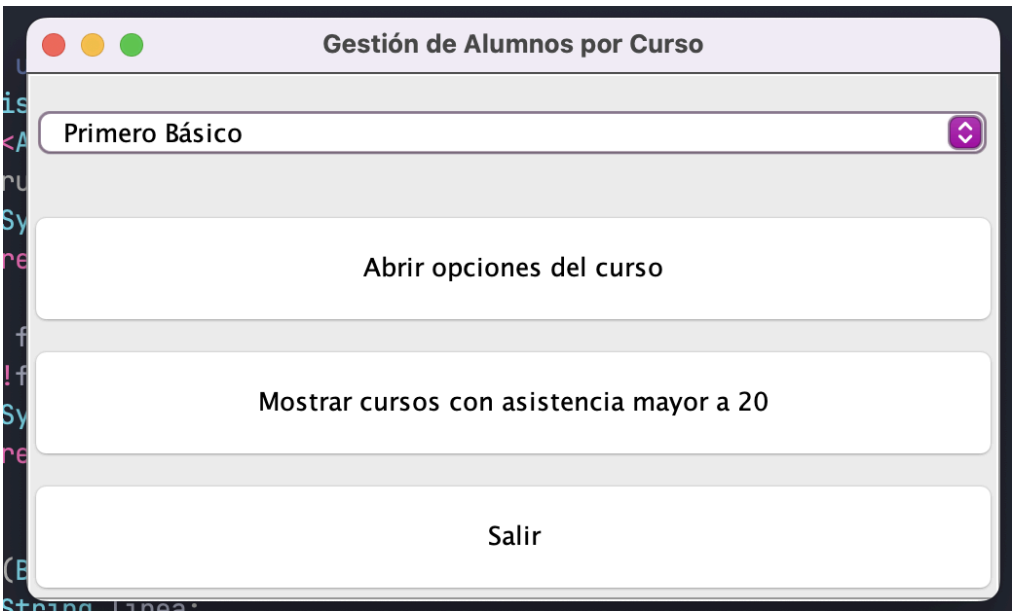
- **PROYECTOSIA1**

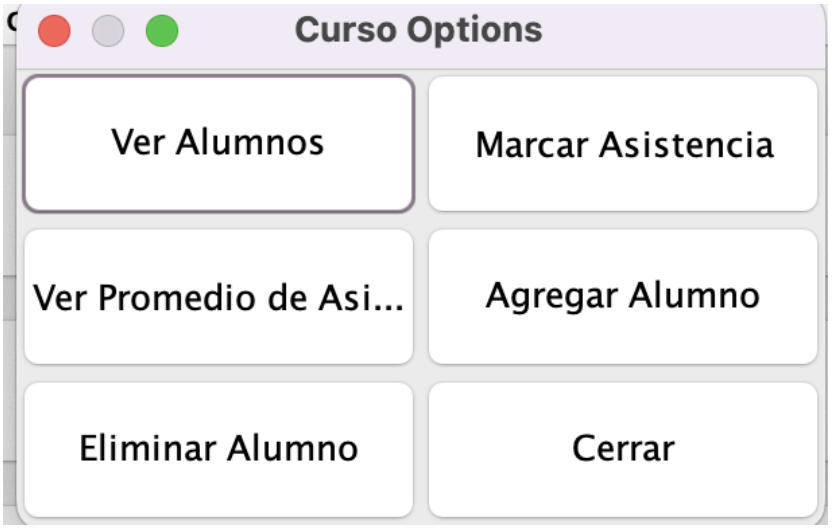


SIA2.2 Persistencia de datos utilizando archivo de texto, CSV, Excel, o conexión con DBMS local (ej. MySQL). Utiliza sistema batch (carga datos al iniciar la aplicación y graba al salir)

Se crearon varios csv, primero para los nombres de los cursos así se puede añadir alguno cuando uno lo necesite sin ningún problema. Además hay csv para cada curso con sus alumnos y otro para la asistencia por día. Cada vez que se ejecuta el programa carga los datos del curso seleccionado, y al salir o cerrar la ventana se guardan.

SIA2.3 La implementación de todas las interfaces gráficas (ventanas) para interactuar con el usuario, considerando componentes SWING.



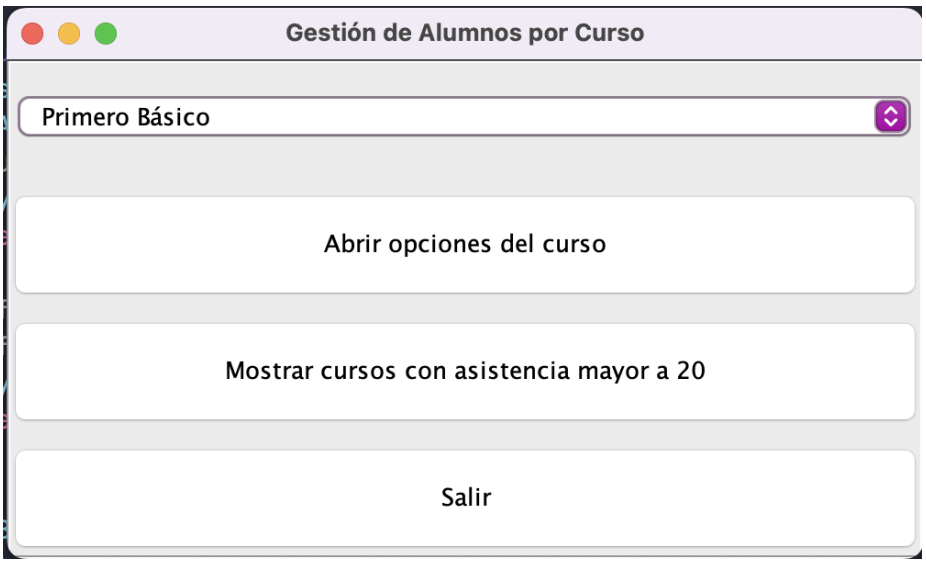


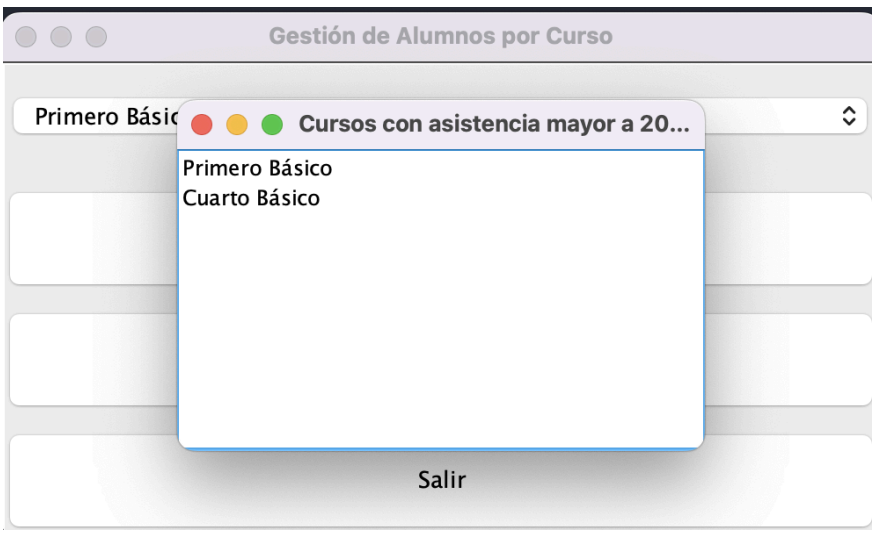
SIA2.4 Se debe hacer un menú para el Sistema donde ofrezca las funcionalidades de: 1) Edición o modificación del elemento y 2) Eliminación del elemento. Esto para la 2a colección de objetos (colección anidada) del SIA1.5

Se puede eliminar o agregar alumno dentro de la lista alumno dentro la colección curso, como se ve en la imagen anterior.

SIA2.5 Se deben incluir al menos 1 funcionalidad propia que sea de utilidad para el negocio (distintas de la inserción, edición, eliminación y reportes).

Se puede filtrar por cursos que tengan asistencia promedio mayor a 20 alumnos, viene por defecto que se filtre primero y cuarto básico.





SIA2.6 El código fuente debe estar bien modularizado de acuerdo a lo descrito en el informe además de seguir las buenas prácticas de documentación interna y legibilidad.

El código está bien documentado y tiene comentarios para mejorar su legibilidad.

SIA2.7 Diseño y codificación de 2 (dos) clases que utilicen sobreescritura de métodos.

En el caso de la clase "**CursoOpciones**", que extiende "**JDialog**", la sobrescritura de métodos se utiliza para personalizar el comportamiento del método "`dispose()`", que es un método de la clase Window en Java Swing utilizado para liberar los recursos asociados a la ventana.

```
public CursoOpciones(JFrame parent, String title, Curso curso) {
    super(parent, title, true);
    this.curso = curso;
    setSize(300, 200);
    setLocationRelativeTo(parent);
    setupUI(curso);

    addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
            dispose();
        }
    });
}

@Override
public void dispose() {
    cerrar();
    super.dispose();
}
```

Para la clase **PROYECTOSIA1**, el código utiliza una forma de sobrescritura implícita al definir una implementación anónima de la interfaz `Runnable` para personalizar el método `run()`.

```
public class PROYECTOSIA1 {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                MainFrame mainFrame = new MainFrame();
                mainFrame.setVisible(true);
            }
        });

    }
}
```

SIA2.8 Implementar el manejo de excepciones capturando errores potenciales específicos mediante Try-catch.

Uso de Bloques Try-Catch para Operaciones de I/O:

- Se implementan bloques try-catch en múltiples métodos para manejar excepciones IOException que pueden surgir durante la lectura y escritura de archivos. Por ejemplo, en los métodos **cargarAlumnosDesdeCSV**, **actualizarCSV** y **actualizarAsistenciasCSV**, se utilizan estos bloques para capturar errores que puedan ocurrir al acceder a los archivos de datos.

Implementación de Excepciones Personalizadas:

- Se utilizan excepciones personalizadas para manejar situaciones específicas más eficazmente. Como es el caso de **ConexionFallidaException** y **ValorInvalidoException.**

Propagación de Excepciones:

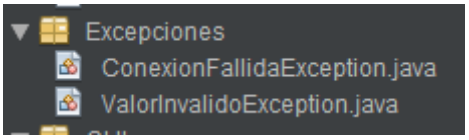
- En varios métodos, las excepciones son propagadas hacia arriba en la pila de llamadas usando el modificador throws en la declaración del método. Esto permite que el manejo de excepciones pueda ser realizado en niveles superiores del programa, lo que es una práctica adecuada en situaciones donde el contexto más amplio es necesario para decidir cómo manejar el error.

Manejo de Errores de Configuración y Datos:

- En el método **obtenerRutaArchivoAsistencia** y otros similares, se verifica la existencia de la ruta del archivo y se lanzan excepciones personalizadas si no se encuentran, lo cual es crucial para evitar errores de tiempo de ejecución y proporcionar retroalimentación específica al usuario o a otros componentes del sistema.

SIA2.9 Crear 2 clases que extiendan de una excepción y que se utilicen en el programa.

Se genera una carpeta de excepciones donde se crean 2 clases que extienden de una excepción.



ConexionFallidaException

Esta excepción se extiende de la clase base Exception y se diseñó para manejar errores relacionados con problemas de conectividad o acceso a archivos. La implementación de esta clase es sencilla, extendiendo Exception y aceptando un mensaje que detalla el problema específico.

```
package Excepciones;

public class ConexionFallidaException extends Exception {
    public ConexionFallidaException(String mensaje) {
        super(mensaje);
    }
}
```

- **Uso en el programa:**
  - **actualizarCSV:** Este método en la clase GestorCSV utiliza ConexionFallidaException para lanzar un error cuando no se puede encontrar la ruta del archivo CSV correspondiente al curso. Si la ruta es null, se lanza la excepción indicando que no se pudo encontrar el archivo.
  - **actualizarAsistenciasCSV:** Similar al método anterior, este método lanza ConexionFallidaException si no se encuentra la ruta del archivo de asistencia para el curso. Además, se utiliza para manejar errores de I/O durante la lectura y escritura del archivo de asistencia, lanzando la excepción con un mensaje que especifica el problema ocurrido durante estas operaciones.

ValorInvalidoException

Esta excepción también se extiende de Exception y se utiliza para manejar errores específicos relacionados con la conversión de datos, como la conversión de un string a entero que falla debido a un formato incorrecto.

```java
package Excepciones;

public class ValorInvalidoException extends Exception {
    public ValorInvalidoException(String mensaje) {
        super(mensaje);
    }
}
```

Uso en el programa:

- **asistenciaHistorica:** En este método de GestorCSV, ValorInvalidoException se lanza cuando se intenta convertir un string a entero y el proceso falla debido a un formato inválido del string. Esta excepción se lanza dentro de un bloque catch que captura NumberFormatException, reemplazando la excepción original con una más descriptiva y específica.

SIA2.10 Se debe generar un reporte en archivo txt que considere mostrar datos de la colección de objetos (ej: csv).

El programa escribe a los alumnos en un archivo csv, vienen ya incorporados pero se pueden eliminar a través de la aplicación.

SIA 2.11 Se implementa el patrón Modelo-Vista-Controlador (MVC) en la arquitectura del sistema.

Si se cumple, ya que algunas clases del package app son de tipo modelo, además del gestor csv es controlador. Vista son las clases MainFrame, ClaseOpciones y SpinnerDatePicker.

- **SIA2.12 Utilización de herramientas de gestión:**

  Recursos Usados:

  - **JDK 17.**
  - **Netbeans 12.**
  - **ChatGPT.**
  - **Trello**
  - **Git y Github.**

2.12.1 GitHub: Utilización de GitHub (Realización de al menos 3 Commit).

https://github.com/sebacruz1/ProyectoPA.git

2.12.2 Trello: Definir y administrar tareas (Completar al menos 5 tareas)

https://trello.com/b/yZG2JbEb/gestion-de-asistencia