Escola Tècnica Superior d'Enginyeria

Universitat Rovira i Virgili

Pràctica 3

Assignatura: Programació

NÚMERO D'EQUIP: 34

EQUIP: Alfonso Sánchez Ferrer

Eduard Vericat Batalla

Carlos Castañón García

Clara Puig Labrador

DIRECTOR: Iván Morillas Gómez

DATA: 23 / 12 / 2023

Índice

1. INTRODUCCIÓN 1

1.1 DISTRIBUCIÓN DE TAREAS 1

1.2 SINCRONIZACIÓN 3

2. ANÁLISIS Y DISEÑO DEL PROGRAMA 5

2.1 ANÁLISIS DEL ENUNCIADO 5

2.2 DECISIONES DE DISEÑO 8

3. EVALUACIÓN DEL PROGRAMA 10

3.1 PRUEBAS REALIZADAS 10

3.2 RESULTADOS OBTENIDOS 10

3.3 VALIDACIÓN DE LAS PRUEBAS 10

4. CONCLUSIONES 11

4.1 INDIVIDUALES 11

1. Introducción

1.1. Distribución de tareas

Terminal:

Para facilitar la distribución de tareas, se han asignado los siguientes roles en el equipo (sin tener en cuenta el orden de desarrollo de las tareas):

- Prog1: Alfonso Sánchez Ferrer
    - Alfonso se encargó de realizar todo el sistema de ficheros y los casos 1-4.
    - Excepciones.
- Prog2: Eduard Vericat Batalla
    - Eduard se encargó de la clase *entitat* y *usuari*, además de los casos 5-8.
- Prog3: Carlos Castañón García
    - Carlos realizo la clase *activitat* y sus subclases (*visita*, *taller* y *xerrada*). También los casos del 9-11.
    - Excepciones.
- Prog4: Clara Puig Labrador
    - Clara programo todas las clases de listas y la clase *reserva*, además de los casos 12-15 (15 sincronizado con prog1).

Respecto al orden de tareas se ha seguido el siguiente orden de ejecución:

1. Clases:
    a. Tareas programador 2.
    b. Tareas programador 3.
    c. Tareas programador 4.
2. Ficheros:
    a. Tarea ficheros programador 1.
3. Casos:
    a. Todos los programadores, cualquier orden.
4. Excepciones:
    a. Todos los programadores se encargaron de verificar que los datos introducidos o el trato de estos no generase excepciones, y los programadores 1 y 3 se encargaron de hacer una revisión final para verificar que todo estaba correcto.

Gráfico:

Con el objetivo de extender los conocimientos en el manejo grafico con Swing, se ha realizado el apartado grafico en todos los casos de la práctica, a parte de la tarea indicada en el manual de la práctica. Para trabajar la parte gráfica solicitada en el manual, los miembros del equipo se han sincronizado. Por otro lado, se ha combinado la consola con el grafico, permitiendo al inicio seleccionar el tipo de ejecución. Además, el control de excepciones en este lo ha realizado cada programador en su caso correspondiente, excepto en el caso 0 donde se ha trabajado en equipo. Los programadores 1 y 3 han verificado nuevamente todo el trabajo, para evitar en la medida de lo posible las excepciones en la aplicación.

Tareas realizadas por orden y ejecución de los programadores:

- **CASO 0:**
  - Este caso es el requerido por la práctica, y ha sido realizado por todos los programadores, distribuyendo el trabajo de la siguiente forma:
    - Prog 1:
      - Modificación de los botones de tipo de actividad, para que si no existe una actividad aparezca en rojo. Ventana donde aparecen los botones con el numero de actividades en un día concreto.
    - Prog 2:
      - Mostrar la información en tablas en modo “popup”.
    - Prog 3:
      - Realización estructura básica de ventana y botones básicos. Función que aplica el nombre del botón al numero de actividades en el día concreto.
    - Prog 4:
      - Verificación de funcionamiento y realización de toda la ventana interior (dentro de un día concreto), excepto el marcado en rojo de los botones donde no existía actividad a mostrar.

- **Resto de casos:**
  - Prog 1:
    - Verificación de los casos.
    - Mejora de diseño en los casos 3, 4, 5, 6, 7, 9, 10.
    - Realización de los casos 8, 11, 12, 13.
    - Control de excepciones (mismo trabajo que en terminal).
  - Prog 2:
    - Implementación de uso de clases con los gráficos.
    - Realización de los casos 1, 2 (son casos de mucho trabajo, sobre todo el 1).
  - Prog 3:
    - Realización de los casos 3, 4, 5, 6, 7, 9, 10.
    - Control de excepciones (mismo trabajo que en terminal).
  - Prog 4:
    - Realización caso 1 en conjunto con prog 2 al ser un caso muy extenso.
    - Realización caso 14.

1.2. Sincronización

Ha resultado esencial mantener una buena organización, por lo que nuestro equipo ha realizado reuniones semanales (al principio de 2h por semana, más adelante entre 4 y 10 horas) a las que asistían todos los miembros. Además, cada miembro ha avanzado individualmente la práctica, siempre intentando conservar cierto nivel de coordinación con el resto del equipo. El trabajo realizado en las reuniones se resume de la siguiente manera:

- Reunión 1: Se habló de la organización del proyecto y se diseñó un esquema desde el que partir para hacer el código
- Reunión 2: Nuestro director se presentó, asignó los roles y explicó la distribución de tareas que tenía pensada para la primera entrega de la práctica
- Reuniones 3-4: Se continuó implementando la consola y corrigiendo errores.
- Reuniones 5-7: Se corrigieron los últimos errores del código de la consola
- Reuniones 8-9: Se terminó de implementar toda la parte gráfica

Por otro lado, más allá de las reuniones de todo el equipo acordadas, se han realizado algunas reuniones telemáticas en las que se han resuelto dudas acerca del proyecto.

2. Análisis y diseño del programa

2.1. Análisis del enunciado

En la primera reunión, se analizó detenidamente el enunciado de la práctica y se elaboró un esquema sobre el funcionamiento previsto del programa:

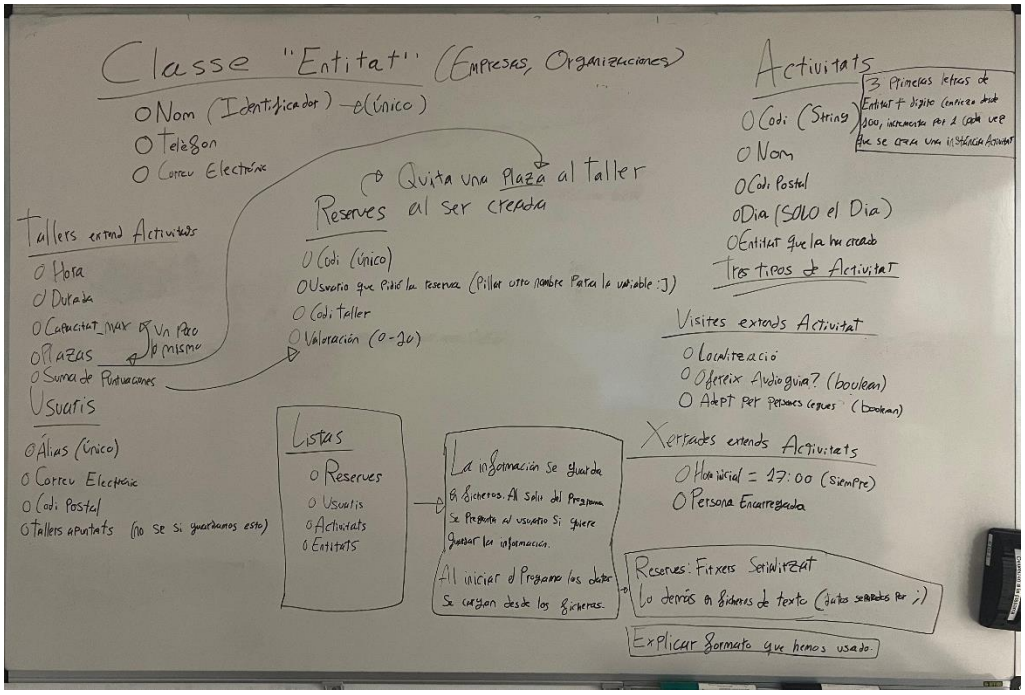

Figura 1: esquema de clases y atributos

*Nota: Las clases aparecen subrayadas en el esquema, los atributos están debajo de cada clase.

A continuación, se concretan los apartados del enunciado en el que se basa la creación de las clases con sus correspondientes atributos:

- Entitat → "Entitats De cadascuna, en guardarem el nom (que l'identifica), telèfon i correu electrònic de contacte. Tindrem una llista d'entitats."
    - Atributos: nombre (nom), número de teléfono (telefon), correo electrónico (correuElectronic)

- Activitat → "Totes les activitats tenen un codi que els identifica, que es genera automàticament a partir de les 3 primeres lletres del nom de l'entitat, i seguides d'un número (començant per 100). De totes les activitats es guarda també el nom, el lloc, el codi postal, i el dia en que es fa (el mes i l'any no cal, ja assumim que són de novembre de 2023). També haurem de poder indicar quina entitat l'ha creat. Per a que hi hagi varietat, les activitats no es repeteixen mai, és a dir, cada activitat ha de guardar un sol dia."
    - Atributos: número que aparece en el código identificador (codiNum), código identificador (codi), nombre de la actividad (nom), código postal (codiPost), día (dia), nombre de la entidad (nomEntitat)

- Visita → "Les visites consisteixen en obrir algun edifici o espai de la ciutat per accés lliure i gratuït. Com unes "portes obertes". Algunes ofereixen audioguia i altres no. Algunes estan adaptades per persones cegues."
    - Atributos: todos los atributos de la clase *activitat*, localización (localitzacio), si tiene o no tiene audioguía (audioguia), si tiene o no adaptación para personas ciegas (adaptPerCecs)

- Taller → "Els tallers es fan en una hora concreta del dia, i tenen una durada determinada. Tenen també una capacitat fixada, i els usuaris s'hi ha de registrar."
    - Atributos: todos los atributos de la clase *activitat*, hora (hora), duración en minutos (durada), capacidad máxima (capacitatMax), número de inscripciones (numInscripcions), puntuación total dada por los usuarios (sumaPuntuaciones), número total de puntuaciones recibidas (numPuntuaciones)

      *Nota: Los dos últimos atributos se basan en un párrafo posterior del enunciado, "Quan l'usuari ha fet el taller, podrà valorar el seu nivell de satisfacció amb un valor en el rang [0-10]. La valoració es guarda en la reserva i s'aplica al taller on es guarda la suma de puntuacions que ha rebut el taller, junt amb el nombre de persones que l'han puntuat."

- Xerrada → "Les xerrades es fan sempre a la mateixa hora, les 17 hores. Volem guardar el nom de la persona que fa la xerrada."
    - Atributos: todos los atributos de la clase *activitat*, nombre de la persona que da la charla (nomPersona), hora (hora, es *static* porque siempre será a las 17)

- Usuari → "Per poder participar en un taller s'han d'inscriure prèviament. Un usuari es dona d'alta al sistema amb un àlies que l'identifica i és diferent per a tots els usuaris. De l'usuari guardem aquest àlies, la seva adreça de correu electrònic i el codi postal d'on viu. Un usuari es pot apuntar a diferents tallers."
    - Atributos: alias (alies), dirección de correo electrónico (correuElectronic), código postal (codiPost), número de inscripciones en un taller (numInscripciones)

      *Nota: El último atributo se ha añadido para controlar que el usuario no se apunte en ningún taller que ya haya alcanzado su capacidad máxima.

- Reserva → "Quan un usuari demana anar a un taller es comprova que en el taller encara hi ha places disponibles i, si és el cas, es genera una instancia reserva amb un codi. De cada reserva es guarda, el codi de la reserva, l'usuari que fa la petició i el codi del taller al què vol assistir. El taller ha de registrar d'alguna forma que ja té un nou assistent i per tant, una plaça disponible menys. Quan l'usuari ha fet el taller, podrà valorar el seu nivell de satisfacció amb un valor en el rang [0-10]. La valoració es guarda en la reserva i s'aplica al taller on es guarda la suma de puntuacions que ha rebut el taller, junt amb el nombre de persones que l'han puntuat."
    - Atributos: código de reserva (codiReserva), código del taller (codi), taller en el que quiere participar el usuario (taller), usuario que quiere participar en el taller (usuari)

      *Nota: Se han creado un atributo taller y otro usuario para tener toda la información de ambos atributos.

De cada una de las clases, se ha implementado una lista (requerimientos no funcionales del enunciado).

Las clases que almacenan la información de la aplicación se volverán persistentes mediante el uso de ficheros. En cada nueva ejecución del programa, se cargará automáticamente toda la información desde los ficheros, y se crearán e inicializarán los objetos de las clases correspondientes. A lo largo de la ejecución de la aplicación, los datos se modificarán en los objetos en memoria como resultado de las funcionalidades solicitadas por los usuarios. Cuando el usuario decida cerrar la aplicación, se le solicitará que determine si desea guardar toda la información que se encuentra en los objetos en memoria en los archivos, sobrescribiendo el contenido existente. Cabe destacar que, a excepción de la clase *Activitat*, cuya información se guarda en un archivo de texto debido a la variabilidad de sus campos, ya que un archivo CSV requiere una estructura fija de columnas y no se ajusta bien a esta situación, las clases se guardan en archivos CSV, de manera que los datos quedan más organizados.

En cuanto al control de excepciones, en el caso de la interfaz gráfica no ha sido necesario utilizar *try catch*, puesto que con la instrucción *if* el programa ya controla correctamente las excepciones de forma que el código final queda más optimizado. Los únicos casos en los que se ha utilizado *try catch* se encuentran en el main (carga y guardado de archivos), además del caso 5 (comprobar si el dato introducido es un número).

2.2. Decisiones de diseño

Previamente se han comentado algunas de clases creadas y sus atributos. Es importante mencionar que todas las clases son públicas salvo *Activitat*, que es abstracta. Esto se debe a que no se crean objetos de tipo *Activitat* como tal, pero sí objetos de tipo *Visita*, *Taller* o *Xerrada* (subclases de la *Activitat*). Sobre los atributos de esta misma clase, cuenta con dos códigos, *codi* y *codiNum*. El primero se crea a partir de las 3 primeras letras del nombre de la entidad y el segundo código. Este segundo código se trata de un número que empieza por 100 (de ahí que se declare el atributo como *static*). A este número se le va a sumar 1 cada vez que se añada una nueva actividad, de modo que no podrá haber dos códigos *codi* iguales.

Además de todas las clases anteriores, se ha creado la clase *LlegirArxius*, diseñada para leer datos de archivos de texto y devolverlos en forma de cadenas. Los atributos de la clase representan los nombres de los archivos que se utilizarán para leer los datos de usuarios, entidades y actividades.

Todas las clases del programa cuentan con los *getters* y *setters* necesarios (en muchos casos se requiere obtener información de una clase para cumplir con los requerimientos del enunciado), además de algunos métodos comunes como *toString*, copiar un elemento (*copia* o *equals*), añadirlo o borrarlo (en el caso de las listas) o encontrar cierto tipo de elementos (como, por ejemplo, *usuariAmbMesReserves* en *LlistaReserves*, que devuelve el usuario que haya realizado un mayor número de reservas). También hay métodos de tipo *boolean* que determinan si cierta actividad tiene adaptación para invidentes o no, si hay audioguía o no, si un usuario está apuntado a un taller…

Como último detalle a destacar, se ha puesto en práctica el uso de herencias en el código para la clase *Activitat*, cuyas subclases son *Visita*, *Taller* y *Xerrada*. Esto supone que las subclases cuentan con los atributos de la clase, además de los suyos propios. A continuación, se presenta un mapa conceptual en el que se puede ver tanto las clases como sus métodos y atributos.

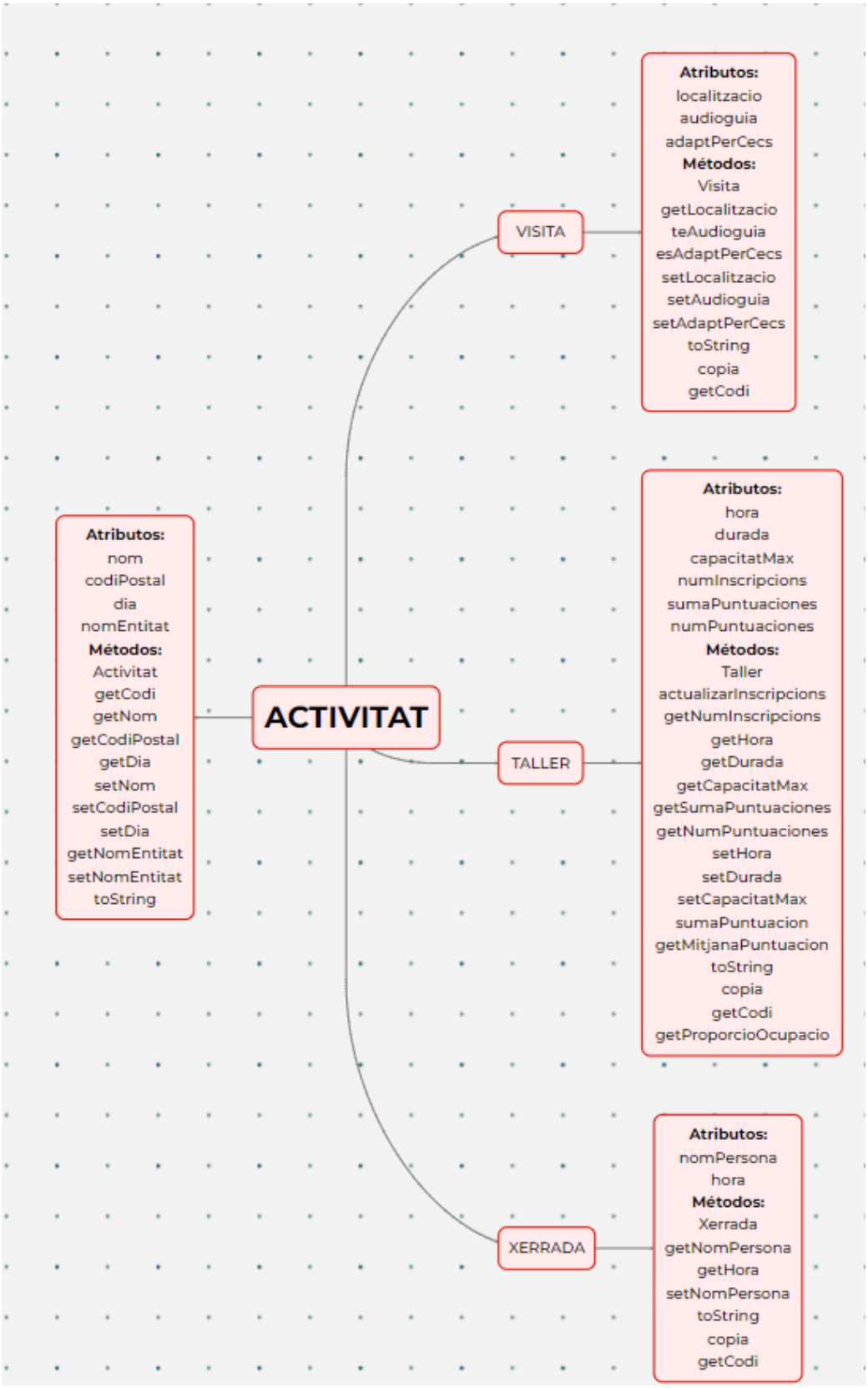

Figura 2: Mapa conceptual de herencias

3. Evaluación del programa

3.1. Pruebas realizadas.

Se han realizado pruebas en cada uno de los casos encontrando fallos generales como:

- Introducir datos fuera de lo esperado. Ejemplo: introducir carácter cuando se espera un número.
- Introducir nombres/códigos de entidades, usuarios, actividades no existentes.
- Introducir datos fuera de rango. Ejemplo: Indicar que la capacidad máxima de un taller es -10.

3.2. Resultados obtenidos.

Errores obtenidos:

- Excepción *NumerFormatException.*
- Datos incoherentes.

3.3. Validación de las pruebas.

Se ha realizado un control de excepciones mediante el uso de instrucciones *if* y bloques *try-catch*, tal y como se muestra en la siguiente imagen:

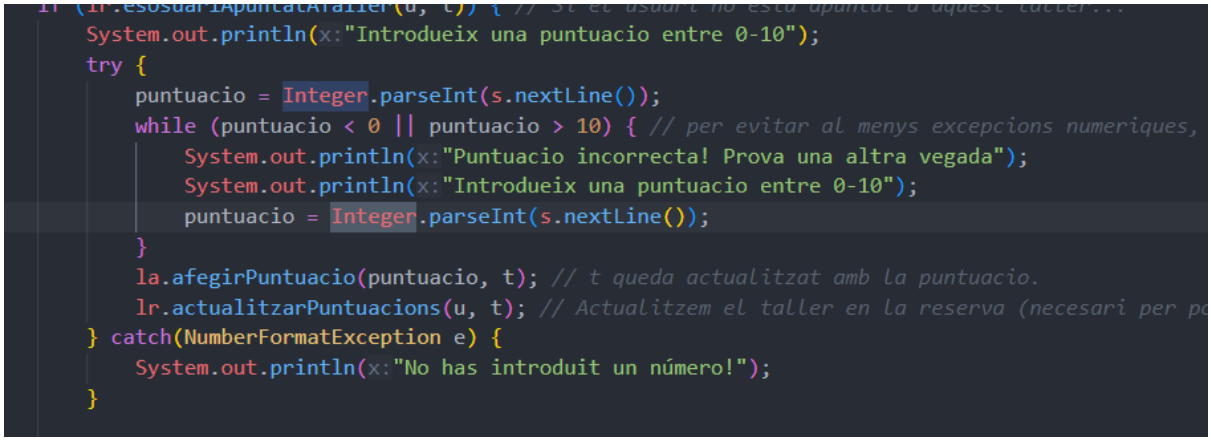

Figura 3: Ejemplo de control de excepciones con *try-catch*

También se generó en el grafico una función que verificaba si era un numero o no:

```
private boolean esNumero(String input) {
    try {
        Integer.parseInt(input);
        return true;
    } catch (NumberFormatException e) { // si no es un numero dor
        return false;
    }
}
```

Figura 4: Ejemplo del control de excepciones con *try-catch*

Esta función, recibe el input del usuario, y retorna true o false según sea un numero o no lo sea, usando las excepciones como calculo para ello.

Al realizar todo esto, obtenemos un código que si el usuario intenta introducir datos inexistentes o inesperados obtenemos la respuesta esperada, en este caso un mensaje de error por pantalla o por terminal, el cual no genera una interrupción inesperada de nuestro programa.

4. Conclusiones

4.1 Individuales

- Alfonso: Considero que mi contribución al proyecto, centrada en el sistema de archivos, ha sido fundamental para establecer una base sólida. Creo que la gestión de excepciones que implementé ayudó a mejorar la aplicación. La colaboración con los demás programadores en el caso 0 fue enriquecedora y permitió una implementación integral.

- Eduard: Creo que mi papel en el desarrollo del programa, al igual que el de mis compañeros, fue crucial para la estructura general del programa. La implementación de los gráficos fue un desafío, más que otras tareas, pero al final fue posible superarlo. Trabajar en conjunto con Prog1 en el caso 0 demostró la importancia de la colaboración para abordar tareas complejas.

- Carlos: Considero que mi parte del trabajo aportó funcionalidades específicas necesarias para la aplicación. La gestión de excepciones en mi parte del código fue un aspecto crítico para garantizar la estabilidad del programa. La colaboración en el caso 0 con el resto de los compañeros, donde me ocupé de la estructura básica de la ventana y botones, mostró la importancia de la cohesión en el desarrollo.

- Clara: Con esta practica he desarrollado todos mis fundamentos en programación, haciendo hincapié a la herencia de clases y el uso de Swing.

Además de disfrutar el compartir los conocimientos con el resto de los compañeros del equipo y así mejorar nuestro nivel de programación.