INSTITUT DE L'EBRE
TORTOSA

Projecte MP03, UF5

Ismael Semmar Galvez

Introducció

Per aquesta UF5 de MP03, el professor ens ha manat fer un projecte. Aquest sería un POJO en el que ens fa implementar diverses característiques com bé pot ser una super col·lecció, desar les dades, random access memory i altres..

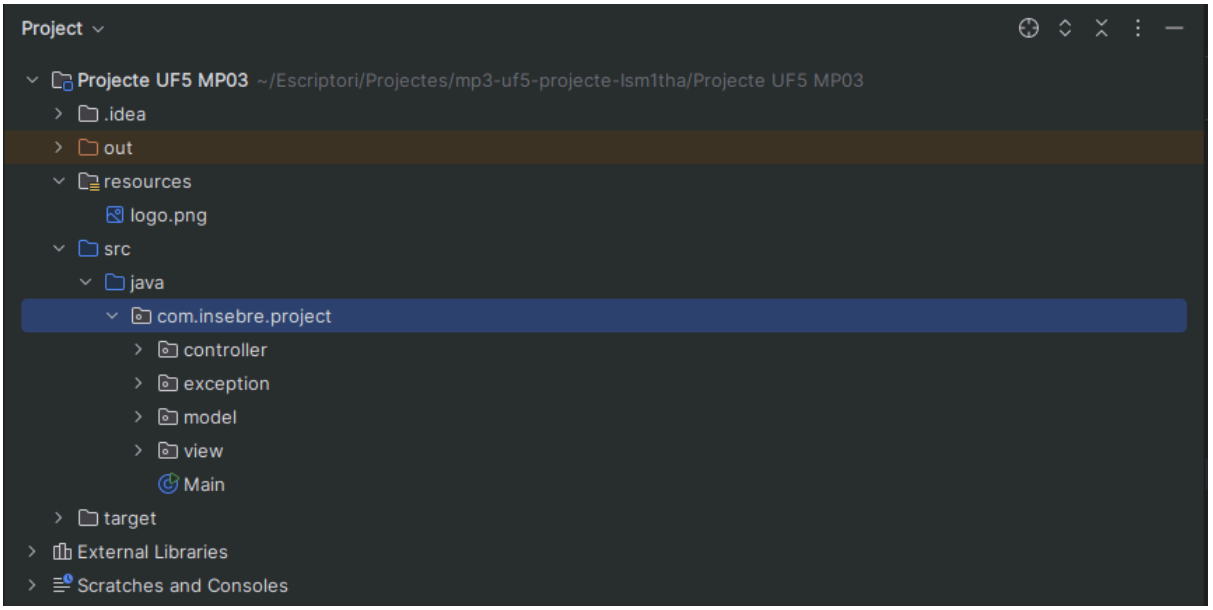
Aniré explicant vista per vista com funciona i el codi que va associat al funcionament d'aquestes.

Vull destacar que el codi està documentat amb javadocs per tant allí també tindrem explica't aquest.

Estructura del projecte

El projecte l'he començat de 0 pero m'he bassat bastant en l'estructura inicial del projecte que proporciona el professor.

Al final l'estructura que proporciona el professor segueix una estructura MVC que es un dels requisits del projecte.



A resources he assignat aquesta com a directori de recursos, en aquest cas per ficar el que vindria a ser el logo de l'aplicació.

Posteriorment podem veure el directori src, podem deduir per el nom que el codi del projecte està allí.
Dintre veurem el directori java i finalment tres paquets fins arribar ara sí al codi font, com.insebre.project.

En aquest punt veurem els següents directoris:

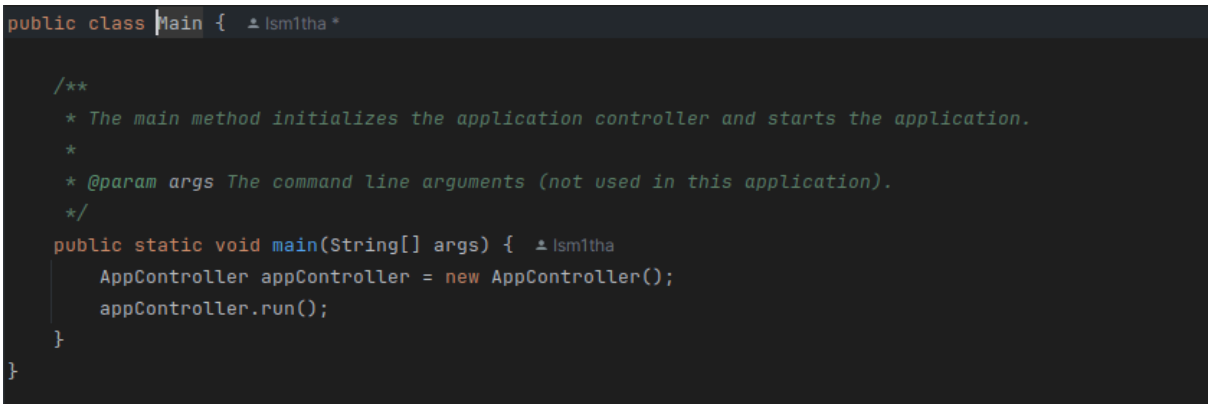
- Controller: Conté tots els controladors de l'aplicació incloent els dels formularis.
- Exception: Conté totes les excepcions personalitzades utilitzades a l'aplicació.
- Model: Els models de l'aplicació, podem veure Program, SuperCollection i version.
- View: A view evidentment tenim les vistes dissenyades amb Swing UI de Java.

Punt d'entrada de l'aplicació

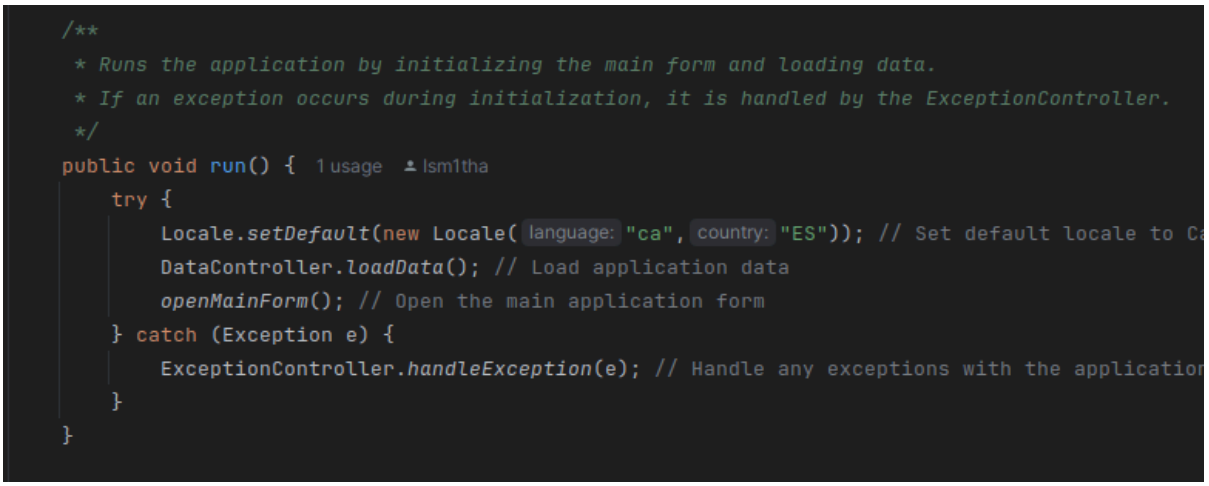
Finalment podem veure el fitxer Main que arranca l'aplicació.

Al fer-ho desde 0, he pensat que seria bona pràctica separar tot el codi en diferents controladors en comptes de tenir-ho tot en un fitxer.

Arranquem l'aplicació per el Main, en aquest fitxer, simplement el que fem es instanciar un objecte de tipus AppController i cridem al mètode run d'aquest.



Al executar run, establim el llenguatge de l'aplicació al català. Posteriorment cridem el mètode que s'encarrega de gestionar les dades del DataController.



Càrrega de dades

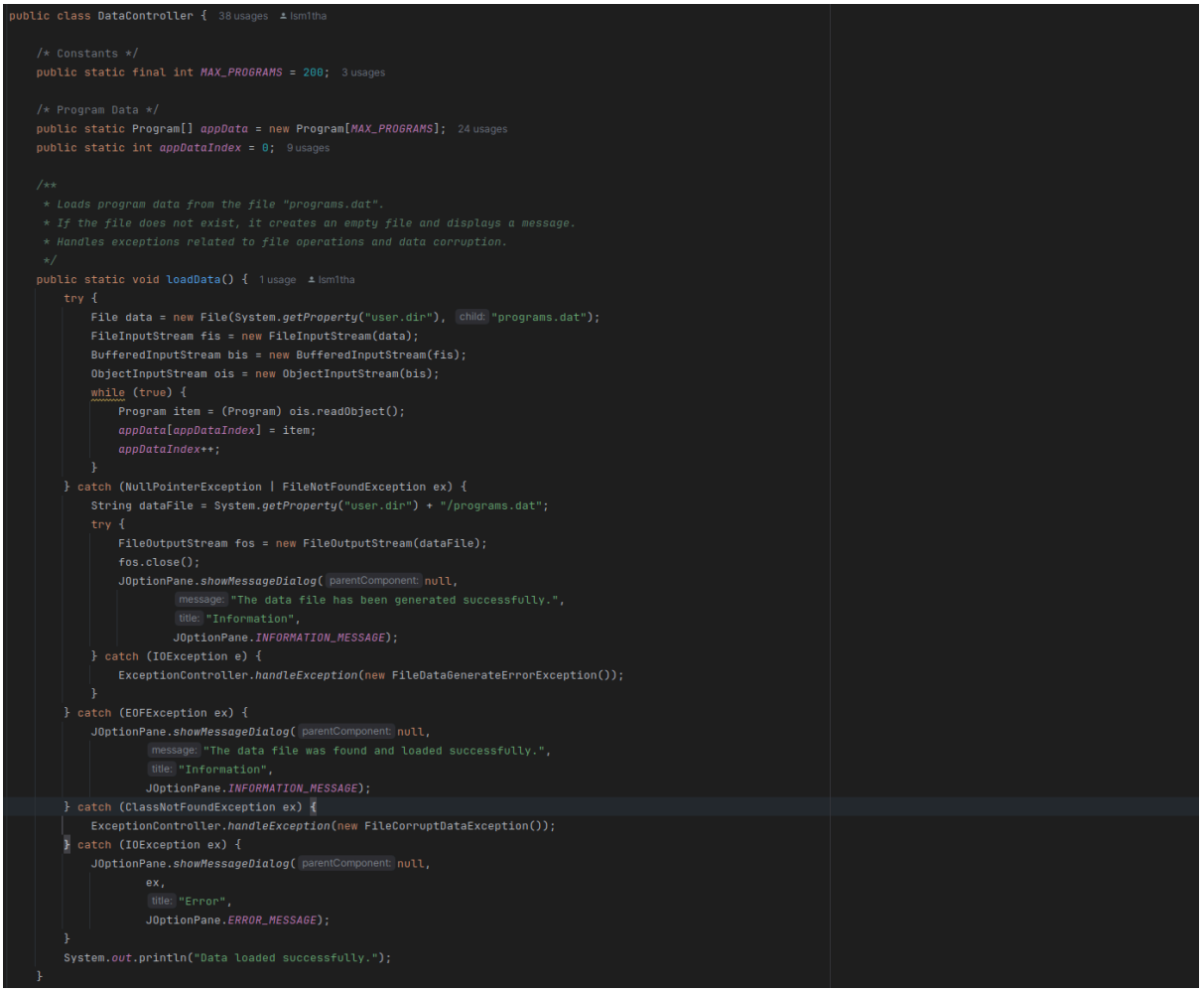
DataController conté tots els mètodes que s'encarreguen de llegir i escriure informació al fitxer de dades.

Els mètodes i variables son estàtics ja que per la forma de treballar i situació em semblava més comoda.

Bàsicament reviso si existeix un fitxer al directori de l'aplicació amb el nom programs.dat. En base a si existeix o no realitzo unes accions o altres (crear, carregar dades al array..).

Defineixo una constant que em delimita el màxim del array de tipo programes on enmagatzemo aquests. El array es de tipo Program, classe que conté la SuperColecció amb versions.

Càrrega de dades:



```
public class DataController {  38 usages   lsm1tha

    /* Constants */
    public static final int MAX_PROGRAMS = 200;  3 usages

    /* Program Data */
    public static Program[] appData = new Program[MAX_PROGRAMS];  24 usages
    public static int appDataIndex = 0;  9 usages

    /**
     * Loads program data from the file "programs.dat".
     * If the file does not exist, it creates an empty file and displays a message.
     * Handles exceptions related to file operations and data corruption.
     */
    public static void loadData() {  1 usage   lsm1tha
        try {
            File data = new File(System.getProperty("user.dir"),  child: "programs.dat");
            FileInputStream fis = new FileInputStream(data);
            BufferedInputStream bis = new BufferedInputStream(fis);
            ObjectInputStream ois = new ObjectInputStream(bis);
            while (true) {
                Program item = (Program) ois.readObject();
                appData[appDataIndex] = item;
                appDataIndex++;
            }
        } catch (NullPointerException | FileNotFoundException ex) {
            String dataFile = System.getProperty("user.dir") + "/programs.dat";
            try {
                FileOutputStream fos = new FileOutputStream(dataFile);
                fos.close();
                JOptionPane.showMessageDialog( parentComponent: null,
                        message: "The data file has been generated successfully.",
                        title: "Information",
                        JOptionPane.INFORMATION_MESSAGE);
            } catch (IOException e) {
                ExceptionController.handleException(new FileDataGenerateErrorException());
            }
        } catch (EOFException ex) {
            JOptionPane.showMessageDialog( parentComponent: null,
                    message: "The data file was found and loaded successfully.",
                    title: "Information",
                    JOptionPane.INFORMATION_MESSAGE);
        } catch (ClassNotFoundException ex) {
            ExceptionController.handleException(new FileCorruptDataException());
        } catch (IOException ex) {
            JOptionPane.showMessageDialog( parentComponent: null,
                    ex,
                    title: "Error",
                    JOptionPane.ERROR_MESSAGE);
        }
        System.out.println("Data loaded successfully.");
    }
```

Estructura de Programa:

```java
import java.io.Serializable;

/**
 * Represents a software program with its details and a collection of versions.
 *
 * @author Ismael Semmar Galvez
 * @version 1.0
 */
public class Program implements Serializable {  22 usages  lsm1tha

    private String name;  3 usages
    private String description;  3 usages
    private String category;  3 usages
    private String language;  3 usages
    private String releaseDate;  3 usages
    private final SuperCollection<Version> versions;  11 usages
```

Control d'excepcions

En aquesta pràctica se'ns va demanar que realitzem un control d'excepcions utilitzant mapping.

En el mètode run de AppController podem veure com ja allí realitzem un try, catch i derivo la excepció al meu ExceptionController.

Veurem que a part d'aquest try catch hi han més per l'aplicació, això és perquè per llançar l'exceció usant Throw ens demana estar dintre d'un bloc try.

```java
public class ExceptionController {  21 usages  lsm1tha

    private static final Map<Class<? extends Exception>, String> exceptionMessages = new HashMap<>();  11 usages

    static {
        exceptionMessages.put(NullPointerException.class, "Null pointer exception occurred");
        exceptionMessages.put(FileNotFoundException.class, "File not found exception occurred");
        exceptionMessages.put(FileNullOnSaveException.class, "No file was found to save the data");
        exceptionMessages.put(FileCorruptDataException.class, "The file data is corrupt");
        exceptionMessages.put(FileDataGenerateErrorException.class, "An error occurred while generating the file data");
        exceptionMessages.put(InvalidPasswordException.class, "Forbidden access. Invalid password");
        exceptionMessages.put(EmptyFieldFoundException.class, "An empty field was found. Please fill all fields");
        exceptionMessages.put(InvalidVersionNameException.class, "Invalid version name");
        exceptionMessages.put(ExistingElementTreeSetException.class, "The element already exists in the TreeSet");
        exceptionMessages.put(InvalidInputFormatException.class, "Invalid input format");
    }

    /**
     * Handles the given exception by building an error message and displaying it using a message dialog.
     *
     * @param ex The exception to handle.
     */
    public static void handleException(Exception ex) {  17 usages  lsm1tha
        String errorMessage = buildErrorMessage(ex);
        showMessageDialog(errorMessage);
    }
```

Simplement creem un objecte exceptionMessages de tipo Map on enmagatzemem una excepció un string que será el missatge de error.

A handleException, cridem buildErrorMessage passant l'excepció i allí ens encarreguem de construir el missatge. En cas de no trobar una excepció retorna Unknown exception i el missatge amb el error.

```java
private static String buildErrorMessage(Exception ex) {  1 usage   lsm1tha
    Class<? extends Exception> exClass = ex.getClass();
    String defaultMessage = "An unexpected error occurred";

    String exceptionMessage = exceptionMessages.getOrDefault(exClass,  defaultValue: "Unknown exception occurred");

    String detailedMessage = ex.getMessage();

    StringBuilder errorMessageBuilder = new StringBuilder();

    if (exceptionMessage != null) {
        errorMessageBuilder.append(exceptionMessage);
    } else {
        errorMessageBuilder.append(defaultMessage);
    }

    if (detailedMessage != null && !detailedMessage.isEmpty()) {
        errorMessageBuilder.append(": ").append(detailedMessage);
    } else {
        errorMessageBuilder.append(".");
    }

    return errorMessageBuilder.toString();
}

/**
 * Displays the given error message in a message dialog with the title "Error".
 *
 * @param errorMessage The error message to display.
 */
private static void showMessageDialog(String errorMessage) {  1 usage   lsm1tha
    JOptionPane.showMessageDialog( parentComponent: null, errorMessage,  title: "Error", JOptionPane.ERROR_MESSAGE);
}
```

Finalment construïm el messagebox amb showMessageDialog.

Formularis

Desde run de AppController, un cop hem establert l'idioma i tenim les dades llestes (fitxer creat o carregat), mostrem el primer formulari.

Aquest primer formulari si que emmagatzemen la instancia en AppController, en la resta no guardo de forma global la instancia dels formularis com aquí. En cas de tancar aquest formulari, l'aplicació es tanca.

```java
/**
 * Opens the main form of the application and initializes its controller.
 * This method sets the table data, loads images, and displays the main form.
 */
public static void openMainForm() {  1 usage  lsm1tha
    mainFormInstance = new MainForm();
    mainFormController = new MainFormController(mainFormInstance);
    mainFormController.setTableData(DataController.getParsedPrograms());
    mainFormController.setImages();
    mainFormController.show();
}
```

Tots els formularis de l'aplicació tenen el mètode show.

Cada formulari conté el seu controlador al directori de Controllers/form i s'instancien de la mateixa forma.

L'única excepció es el formulari de contrasenya que no vaig voler fer un controlador ja que em sembla totalment necessari per la seva funcionalitat i conté molt poc codi dintre de la classe del formulari.

Les classes dels formularis contenen simplement getters per als components i així poder interactuar amb ells desde els respectius controladors amb els listeners.

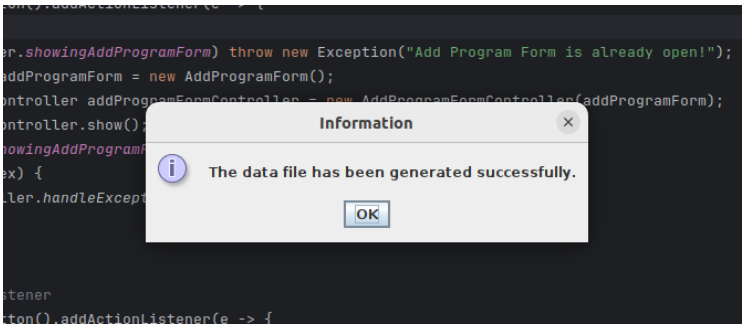
```java
/**
 * Displays the main form.
 */
public void show() {  1 usage  lsm1tha
    JFrame frame = new JFrame( title: "Software Manager - Ismael SG");
    frame.setContentPane(mainForm.getPanel());
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
    mainForm.getTable().setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
}
```

En el constructor es on enmagatzemo el formulari per poder interactuar amb ell, posteriorment es on afegeixo els listeners per a definir les accions que realitzem en els formularis.

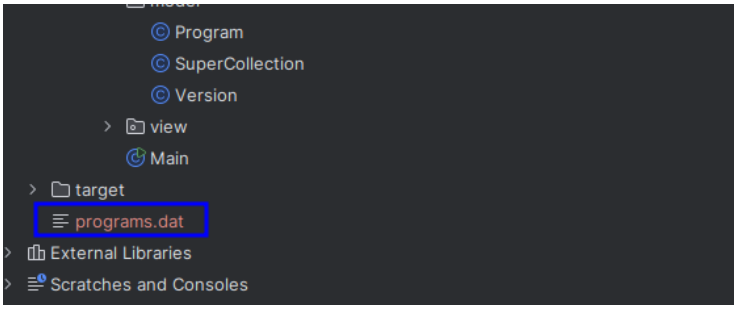
```java
public class MainFormController {  3 usages  lsm1tha

    private final MainForm mainForm;  31 usages

    /**
     * Constructor for MainFormController.
     *
     * @param mainForm The main form instance to be controlled.
     */
    public MainFormController(MainForm mainForm) {  1 usage  lsm1tha
        this.mainForm = mainForm;

        // Add Button ActionListener
        this.mainForm.getAddButton().addActionListener(e -> {
            try {
                if (AppController.showingAddProgramForm) throw new Exception("Add Program Form is already open!");
                AddProgramForm addProgramForm = new AddProgramForm();
                AddProgramFormController addProgramFormController = new AddProgramFormController(addProgramForm);
                addProgramFormController.show();
                AppController.showingAddProgramForm = true;
            } catch (Exception ex) {
                ExceptionController.handleException(ex);
            }
        });

        // Edit Button ActionListener
        this.mainForm.getEditButton().addActionListener(e -> {
            if (mainForm.getTable().getSelectedRow() == -1) {
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Please select a program to edit!",  title: "Error", JOptionPane.ERROR_MESSAGE);
```

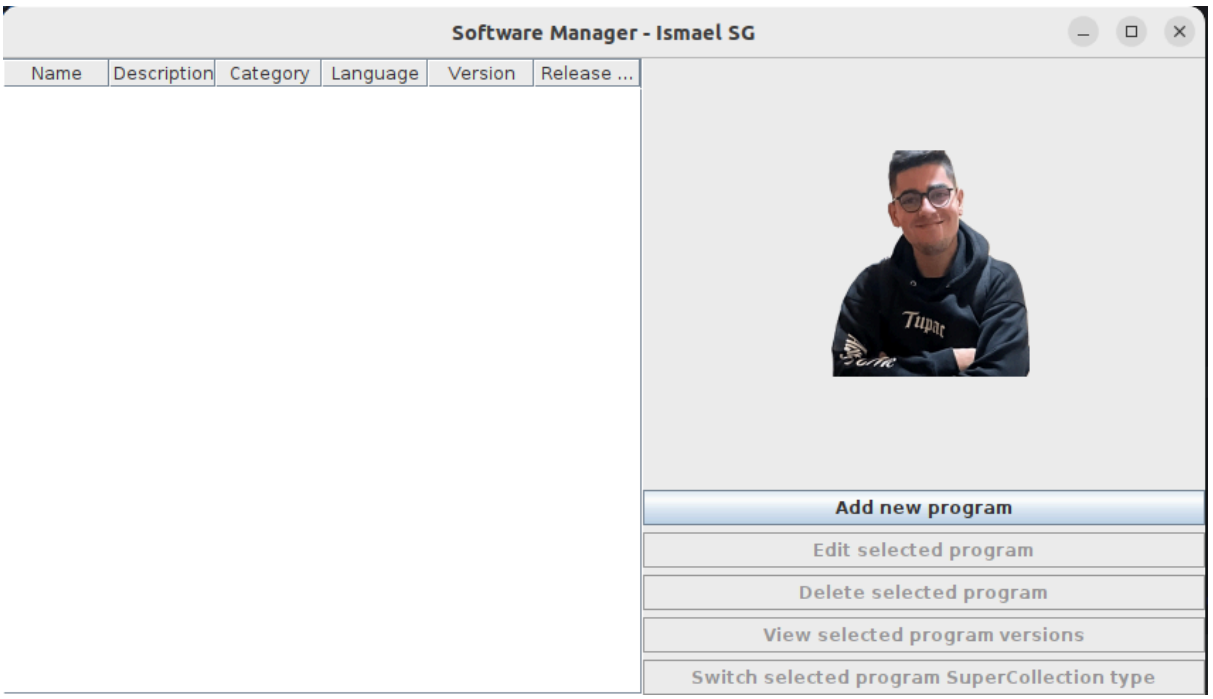
Primer formulari

Com a primer arranc, ens diu que el fitxer de programes ha sigut creat satisfactòriament.



Podem veure el fitxer com s'ha creat correctament a l'arrel del projecte.





De forma predeterminada tenim els botons que ens permeten realitzar accions sobre els diferents registres de la taula.

En fer clic a algun element aquests botons s'activaran.

Els deshabilito desde el constructor del formulari.

```
public MainForm() {  1 usage   lsm1tha
    selectedSuperCollectionTotalVersionsLabel.setVisible(false);
    selectedSuperCollectionTypeLabel.setVisible(false);
    editSelectedProgramButton.setEnabled(false);
    deleteSelectedProgramButton.setEnabled(false);
    viewSelectedProgramVersionsButton.setEnabled(false);
    switchSelectedProgramSupTypeButton.setEnabled(false);
    selectedSuperCollectionLastTotalVersionsLabel.setVisible(false);
}
```

A l'executar el mètode de openMainForm, cridaven el mètode setTableData, aquest mètode assignen la informació passada per paràmetre (programes) a la taula.

```
public void setTableData(Object[][] data) {  2 usages   lsm1tha
    DefaultTableModel model = new DefaultTableModel(data, new String[]{"Name", "Description", "Category", "Language", "Version", "Release Date"}) {  lsm1tha
        @Override  lsm1tha
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    mainForm.getTable().setModel(model);
}
```

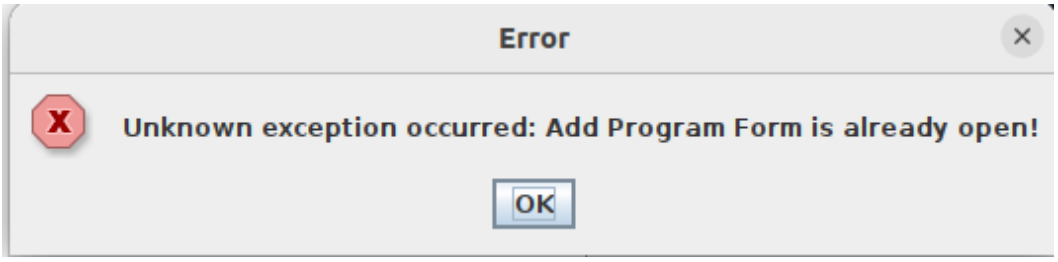
Afegir un nou programa

```
// Add Button ActionListener
this.mainForm.getAddButton().addActionListener(e -> {
    try {
        if (AppController.showingAddProgramForm) throw new Exception("Add Program Form is already open!");
        AddProgramForm addProgramForm = new AddProgramForm();
        AddProgramFormController addProgramFormController = new AddProgramFormController(addProgramForm);
        addProgramFormController.show();
        AppController.showingAddProgramForm = true;
    } catch (Exception ex) {
        ExceptionController.handleException(ex);
    }
});
```

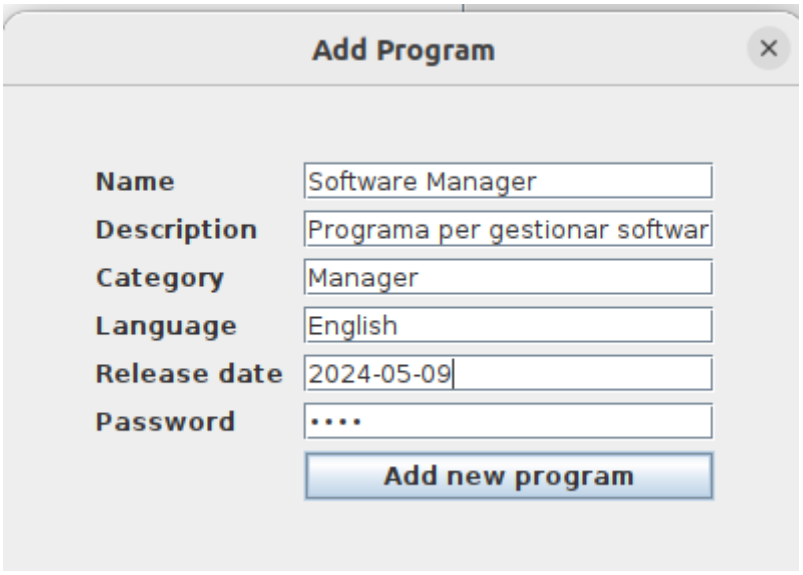
Simplement repetim el que hem fet amb el mainForm per obrir el formulari. Instanciem el formulari i el controlador, passem el formulari de paràmetre a aquest últim.

Podem veurem com defineixo showingAddProgramForm. Tinc aquest tipus de variables estàtiques a AppController per evitar que s'obrin més d'un cop algunes pestanyes.

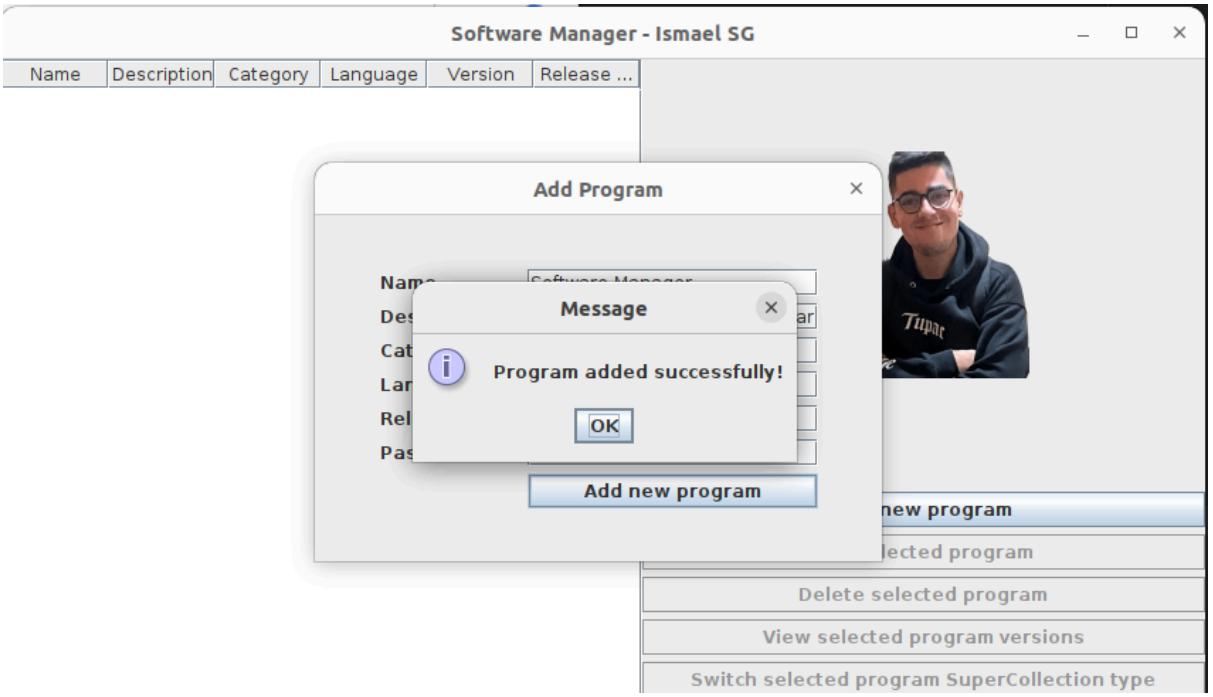
```
/**
 * Flag indicating whether the add program form is currently displayed.
 */
public static boolean showingAddProgramForm = false;  3 usages

/**
 * Flag indicating whether the edit program form is currently displayed.
 */
public static boolean showingEditProgramForm = false;  3 usages

/**
 * Flag indicating whether the view program version form is currently displayed.
 */
public static boolean showingViewProgramVersionForm = false;  4 usages
```



Un cop tenim el formulari obert, simplement omplirem per afegir un nou programa.



Validem els camps desde el listener al fer clic a afegir programa. Ens retornarà en cas de haver un camp buit el camp o per exemple, si la data o la contrassenya no son vàlids també ens avisa.

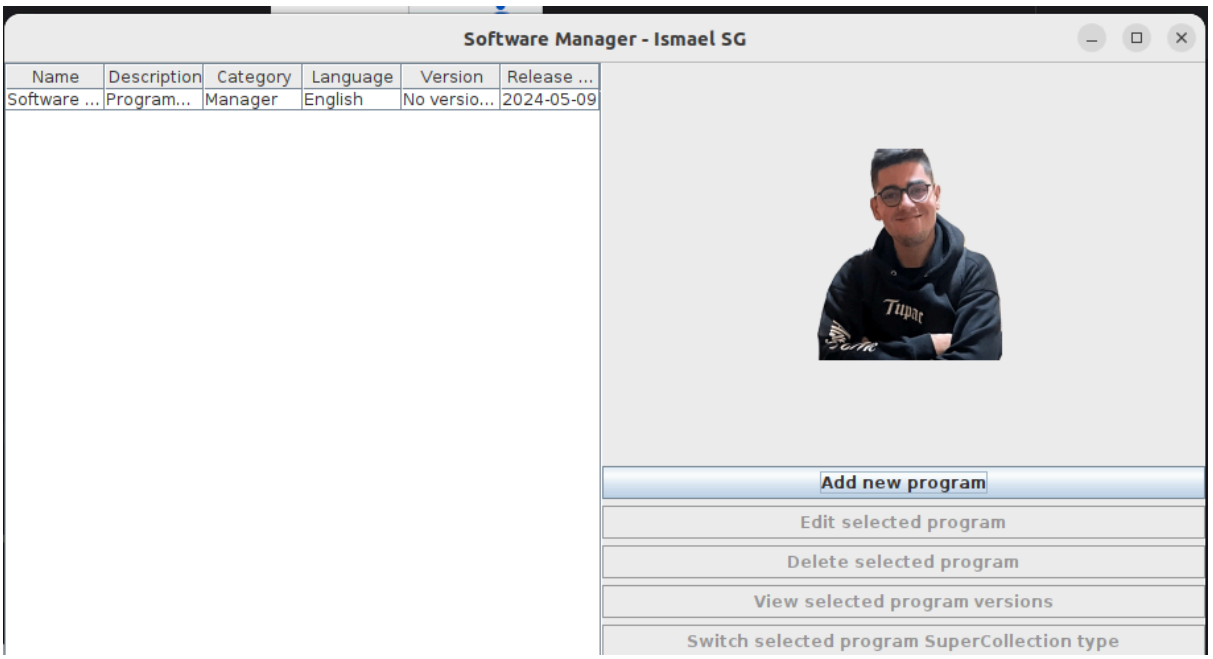


Realitzem simplement les validacions respectives dintre del AddProgramFormController, en cas de estar tot correcte, utilitzant els mètodes per gestionar dades estàtics creats a DataController afegim el nou programa al array de dades.

```
addProgramForm.getSubmitButton().addActionListener(e -> {
        if (AppController.validateReleaseDate(releaseDate)) {
            throw new InvalidInputFormatException("Invalid release date format (yyyy-mm-dd) or future date");
        }

        // Check if maximum program limit is reached
        if (DataController.appDataIndex >= DataController.appData.length) {
            throw new Exception("The maximum number of programs has been reached");
        }

        // Check password length
        if (password.length() != PasswordController.PASSWORD_LENGTH) {
            throw new Exception("The password must be 4 characters long");
        } else {
            // Create a new program instance
            Program program = new Program(name, description, category, language, releaseDate);

            // Add the program to the data array
            Program[] currentData = DataController.getData();
            int currentIndex = DataController.appDataIndex;
            currentData[currentIndex] = program;
            DataController.setData(currentData);
            DataController.appDataIndex++;

            // Save data and insert program password
            DataController.saveData();
            PasswordController.insertProgramPassword(currentIndex, password);

            // Show success message and dispose the dialog
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Program added successfully!");
            dialog.dispose();
        }
    } catch (Exception ex) {
        ExceptionController.handleException(ex);
    }
```

En un parell de vistes, tinc un listener que al tancar la finestra m'executa una acció, en aquest cas crido un mètode que m'actualitza la taula amb les noves dades gràcies a la instancia del mainForm que tenim emmagatzemada.

```
// Window listener to handle form closing event
dialog.addWindowListener((WindowAdapter) windowClosed(e) -> {
        AppController.showingAddProgramForm = false;
        AppController.refreshMainForm();
});
```



El camp version, agafa l'última versió afegida dintre de la Super col·lecció de versions dintre del programa.

RandomAccess

En la pràctica se'ns va demanar utilitzar RandomAccess. El professor ens ha proposat implementar-ho amb un sistema de contrasenyes.

Tinc un controlador per a les contrasenyes amb mètodes estàtics que em permeten llegir i escriure en el fitxer de contrasenyes utilitzant random access.

En el controlador defineixo la mida de les contrasenyes (també afecta a la validació dels formularis).

```
public class PasswordController {  13 usages   Ism1tha
    /* Constants */
    public static final int PASSWORD_LENGTH = 4;  9 usages
```

Agafem l'índex del programa multipliquem per els caràcters i obtindrem la posició inicial dintre del fitxer.

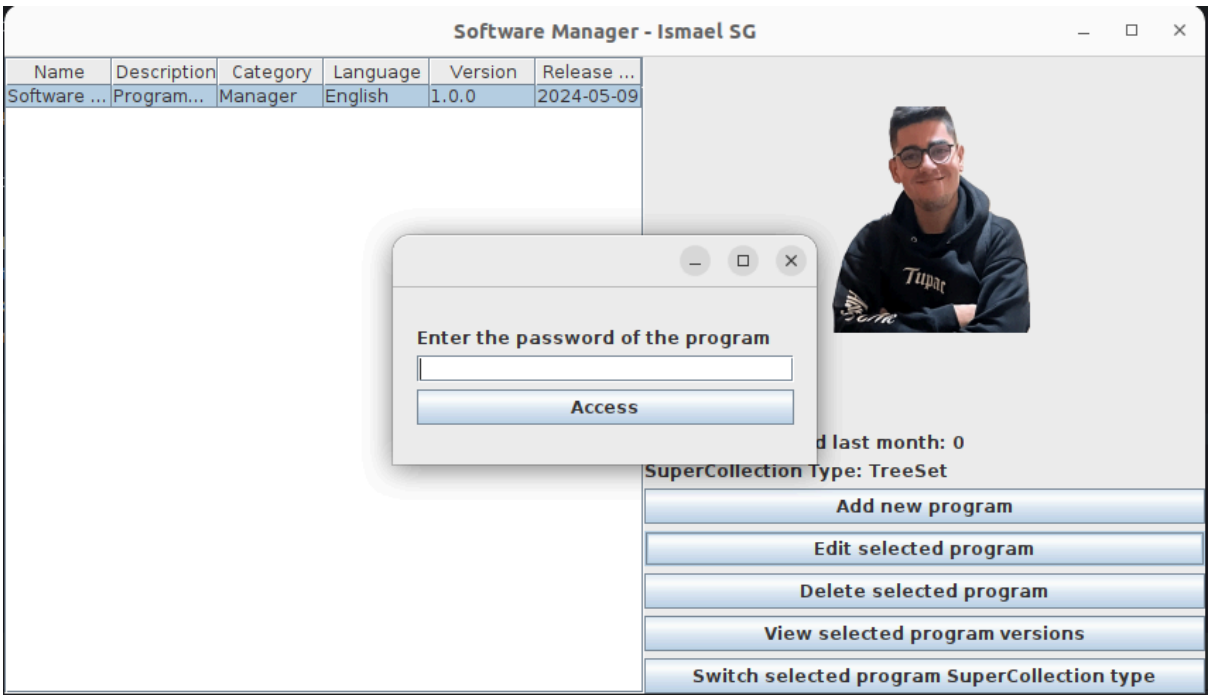
```java
public static void insertProgramPassword(int programIndex, String password) {  2 usages  lsm1tha
    try (RandomAccessFile file = new RandomAccessFile( name: "passwords.dat",  mode: "rw")) {
        int position = programIndex * PASSWORD_LENGTH;
        file.seek(position);
        file.write(password.getBytes());
    } catch (IOException ex) {
        ExceptionController.handleException(ex);
    }
}

/**
 * Reads the password associated with a specific program index from the passwords file.
 *
 * @param programIndex The index of the program for which to read the password.
 * @return The password associated with the program index.
 */
public static String readProgramPassword(int programIndex) {  2 usages  lsm1tha
    String password = null;
    try (RandomAccessFile file = new RandomAccessFile( name: "passwords.dat",  mode: "r")) {
        int position = programIndex * PASSWORD_LENGTH;
        file.seek(position);
        byte[] passwordBytes = new byte[PASSWORD_LENGTH];
        file.read(passwordBytes);
        password = new String(passwordBytes).trim(); // Assuming passwords are null-terminated
    } catch (IOException ex) {
        ExceptionController.handleException(ex);
    }
    return password;
}
```

Quan eliminem una contrasenya regenerem el conjunt de bytes un cop hem eliminat els bytes que ocupen la contrasenya i guardem el fitxer.

```java
public static void deleteProgramPassword(int programIndex) {  1 usage  lsm1tha
    try (RandomAccessFile file = new RandomAccessFile( name: "passwords.dat",  mode: "rw")) {
        byte[] newFile = new byte[(int) file.length() - PASSWORD_LENGTH];
        int position = programIndex * PASSWORD_LENGTH;
        for (int i = 0; i < position; i++) {
            newFile[i] = file.readByte();
        }
        position += PASSWORD_LENGTH;
        file.seek(position);
        for (int i = position; i < file.length(); i++) {
            newFile[i - PASSWORD_LENGTH] = file.readByte();
        }
        file.setLength(0);
        file.write(newFile);
    } catch (IOException ex) {
        ExceptionController.handleException(ex);
    }
}
```

Verificar contrasenya al realitzar accions

En el programa al obrir algunes pestanyes verifico abans que l'usuari està autoritzat demanant-li que introdueixi la contrasenya del programa.



En cas de introduirla correctament ens obrirà la pestanya si no ens llança una excepció.

```
// Edit Button ActionListener
this.mainForm.getEditButton().addActionListener(e -> {
    if (mainForm.getTable().getSelectedRow() == -1) {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Please select a program to edit!",  title: "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    PasswordPromptForm passwordPromptForm = new PasswordPromptForm();
    passwordPromptForm.setVisible(true);
    passwordPromptForm.addWindowListener((WindowAdapter) windowClosed(e) → {
            try {
                if (passwordPromptForm.isPasswordSubmittedSuccessfully()) throw new InvalidPasswordException();
                else {
                    if (PasswordController.checkProgramPassword(mainForm.getTable().getSelectedRow(), passwordPromptForm.getSubmittedPassword()))
                        throw new InvalidPasswordException();
                    if (AppController.showingEditProgramForm) throw new Exception("Edit Program Form is already open!");
                    AppController.showingEditProgramForm = true;
                    EditProgramForm editProgramForm = new EditProgramForm();
                    EditProgramFormController editProgramFormController = new EditProgramFormController(editProgramForm, mainForm.getTable().getSelectedRow());
                    editProgramFormController.show();
                }
            } catch (Exception ex) {
                ExceptionController.handleException(ex);
            }
    });
});
```

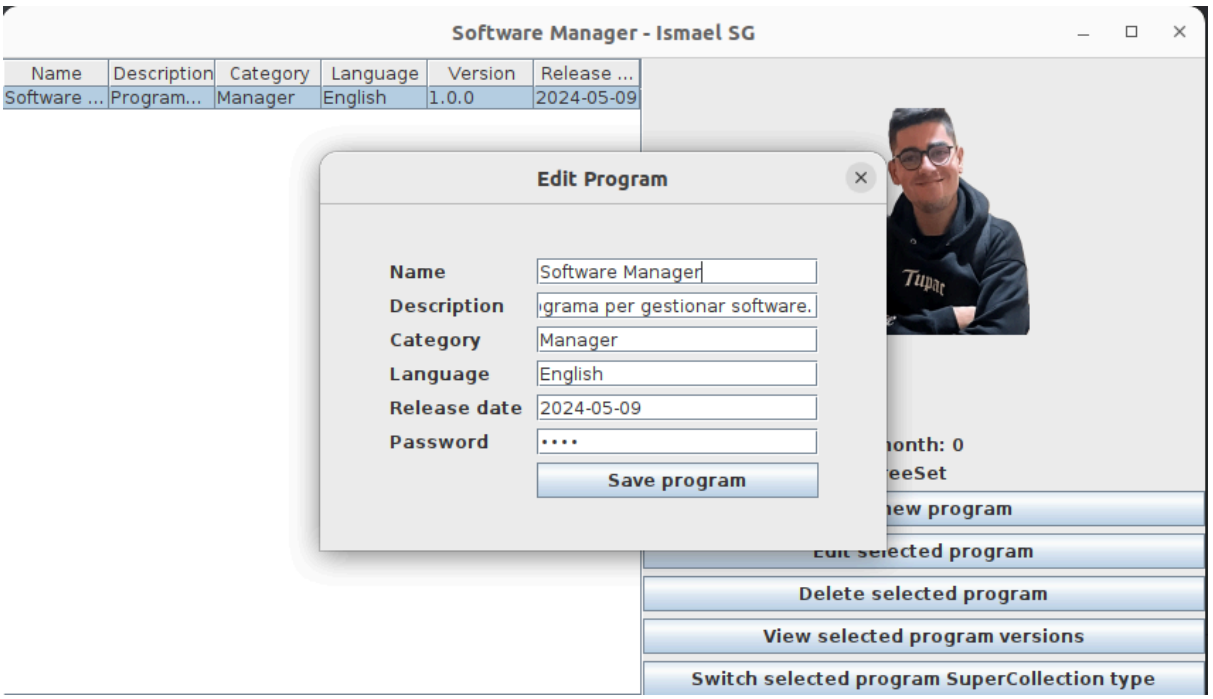
Aquí podem veure el botó de modificar, instanciem el formulari, en aquest cas afegim un listener per quan es tanqui, verificar si l'usuari ha interactuat amb el formulari i ha introduït correctament la contrasenya.

PasswordForm no conté controlador i te aquests pocs mètodes en el formulari per poder setejar el estat i així saber si ha introduït la contrasenya, si es correcta o si ha tancat la finestra.

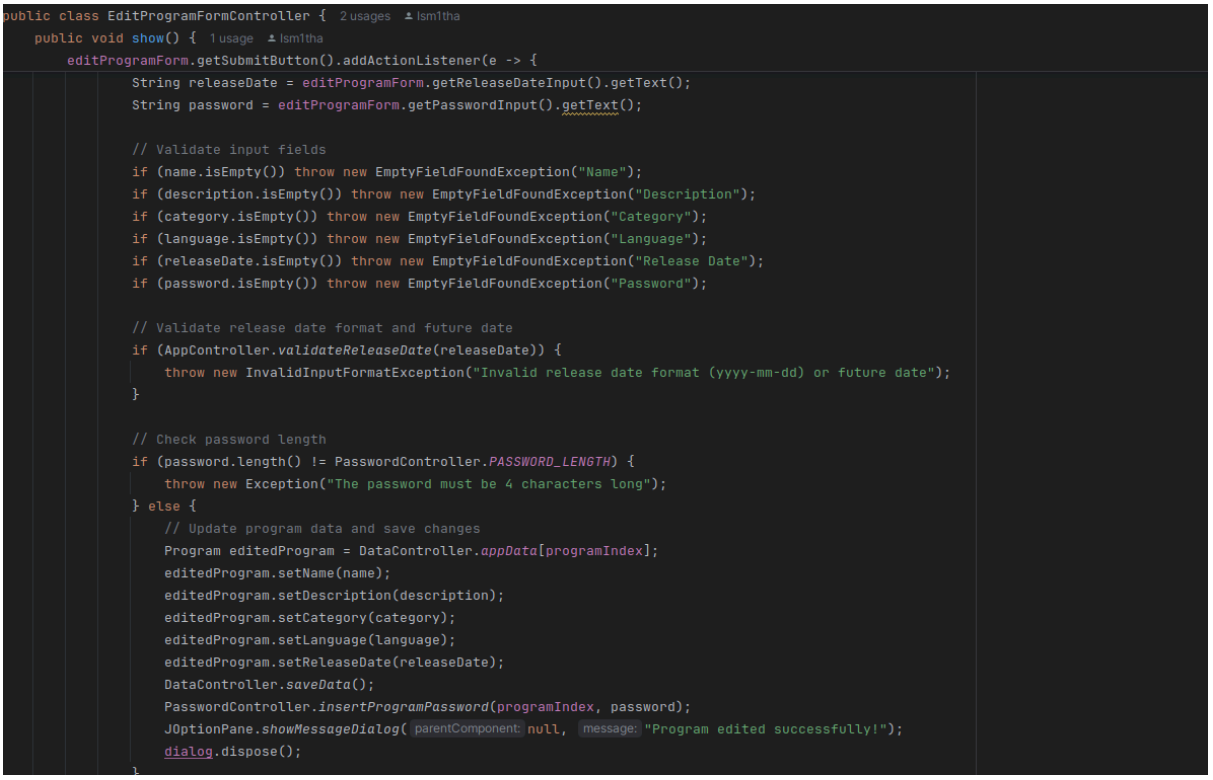
```java
public class PasswordPromptForm extends JFrame{  10 usages  lsm1tha

    private String password;  2 usages
    private boolean isPasswordSubmittedSuccessfully = false;  2 usages

    private JPanel panel1;  2 usages
    private JPasswordField passwordField1;  2 usages
    private JButton accessButton;  2 usages

    public PasswordPromptForm() {  4 usages  lsm1tha

        setContentPane(panel1);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        pack();
        setLocationRelativeTo(null);

        accessButton.addActionListener(e -> {
            String password = String.valueOf(passwordField1.getPassword());
            if(password.length() != 4) {
                closeForm();
            }
            else {
                setPasswordSubmittedSuccessfully(true);
                setSubmittedPassword(password);
                closeForm();
            }
        });
    }

    public void closeForm() { dispose(); }

    public void setPasswordSubmittedSuccessfully(boolean submittedSuccessfully) {  1 usage  lsm1tha
        this.isPasswordSubmittedSuccessfully = submittedSuccessfully;
```

Editar un programa

Simplement ens obre una pestanya igual que la de afegir un programa i ens carrega les dades actuals del programa per modificar-les.

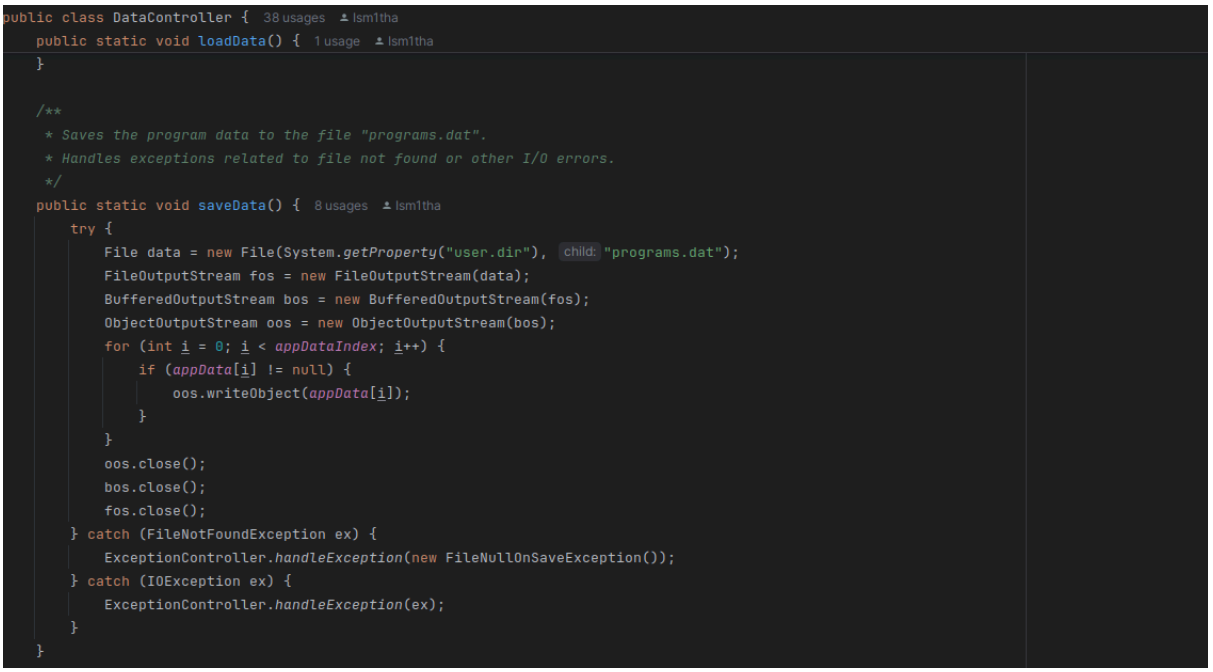


En quant a codi pues bàsicament modifico l'objecte del array i el actualitzo. Destaco que deso cada cop tant al crear, modificar i eliminar al fitxer cridant el mètode estàtic creat a DataController.

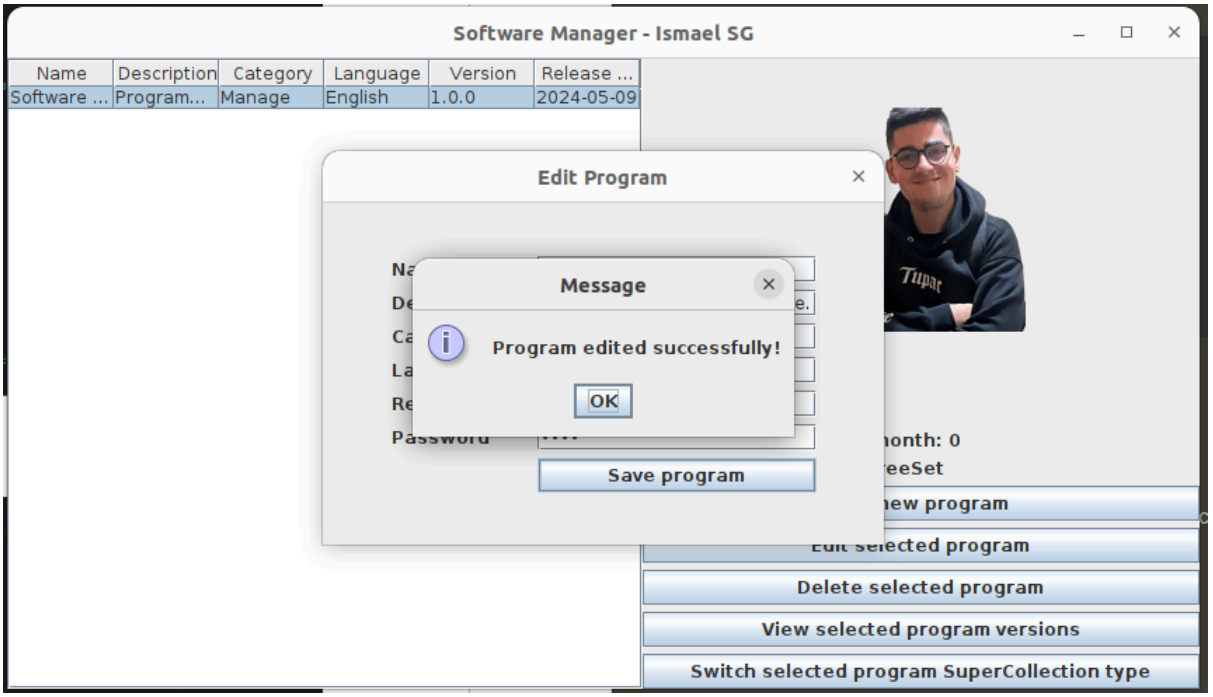


```
public class EditProgramFormController {  2 usages  lsm1tha
    public void show() {  1 usage  lsm1tha
        editProgramForm.getSubmitButton().addActionListener(e -> {
            String releaseDate = editProgramForm.getReleaseDateInput().getText();
            String password = editProgramForm.getPasswordInput().getText();

            // Validate input fields
            if (name.isEmpty()) throw new EmptyFieldFoundException("Name");
            if (description.isEmpty()) throw new EmptyFieldFoundException("Description");
            if (category.isEmpty()) throw new EmptyFieldFoundException("Category");
            if (language.isEmpty()) throw new EmptyFieldFoundException("Language");
            if (releaseDate.isEmpty()) throw new EmptyFieldFoundException("Release Date");
            if (password.isEmpty()) throw new EmptyFieldFoundException("Password");

            // Validate release date format and future date
            if (AppController.validateReleaseDate(releaseDate)) {
                throw new InvalidInputFormatException("Invalid release date format (yyyy-mm-dd) or future date");
            }

            // Check password length
            if (password.length() != PasswordController.PASSWORD_LENGTH) {
                throw new Exception("The password must be 4 characters long");
            } else {
                // Update program data and save changes
                Program editedProgram = DataController.appData[programIndex];
                editedProgram.setName(name);
                editedProgram.setDescription(description);
                editedProgram.setCategory(category);
                editedProgram.setLanguage(language);
                editedProgram.setReleaseDate(releaseDate);
                DataController.saveData();
                PasswordController.insertProgramPassword(programIndex, password);
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Program edited successfully!");
                dialog.dispose();
```

Aquí podem veure el mètode saveData de DataController.



```
public class DataController {  38 usages  lsm1tha
    public static void loadData() {  1 usage  lsm1tha
    }

    /**
     * Saves the program data to the file "programs.dat".
     * Handles exceptions related to file not found or other I/O errors.
     */
    public static void saveData() {  8 usages  lsm1tha
        try {
            File data = new File(System.getProperty("user.dir"),  child: "programs.dat");
            FileOutputStream fos = new FileOutputStream(data);
            BufferedOutputStream bos = new BufferedOutputStream(fos);
            ObjectOutputStream oos = new ObjectOutputStream(bos);
            for (int i = 0; i < appDataIndex; i++) {
                if (appData[i] != null) {
                    oos.writeObject(appData[i]);
                }
            }
            oos.close();
            bos.close();
            fos.close();
        } catch (FileNotFoundException ex) {
            ExceptionController.handleException(new FileNullOnSaveException());
        } catch (IOException ex) {
            ExceptionController.handleException(ex);
        }
    }
```



Eliminar un programa

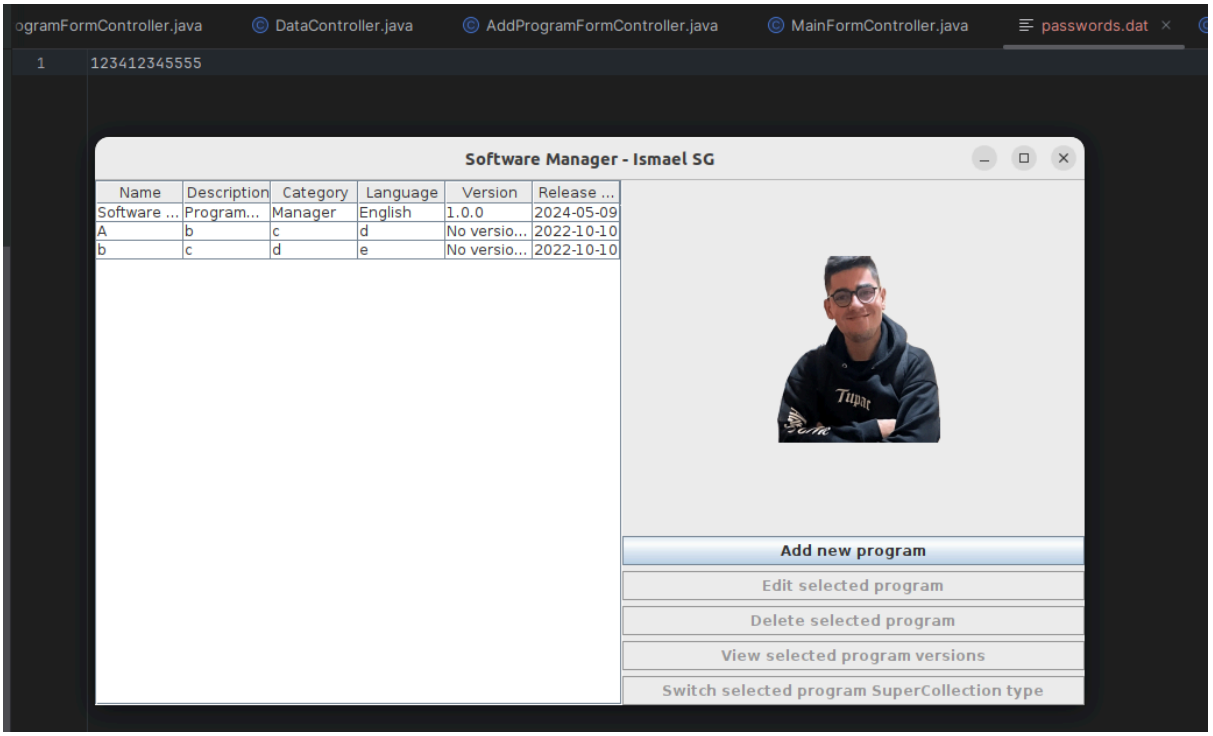
Per eliminar un programa simplement seleccionem el programa i fem clic al botó de delete. Utilitzant el controlador de contrasenyes eliminem aquesta i també eliminem el element del array de programes de l'aplicació.

Un cop he eliminat el programa, tinc un mètode que em desplaça tots els programes a l'esquerra, d'aquesta forma anirà sempre d'acord amb el fitxer de contrasenyes.

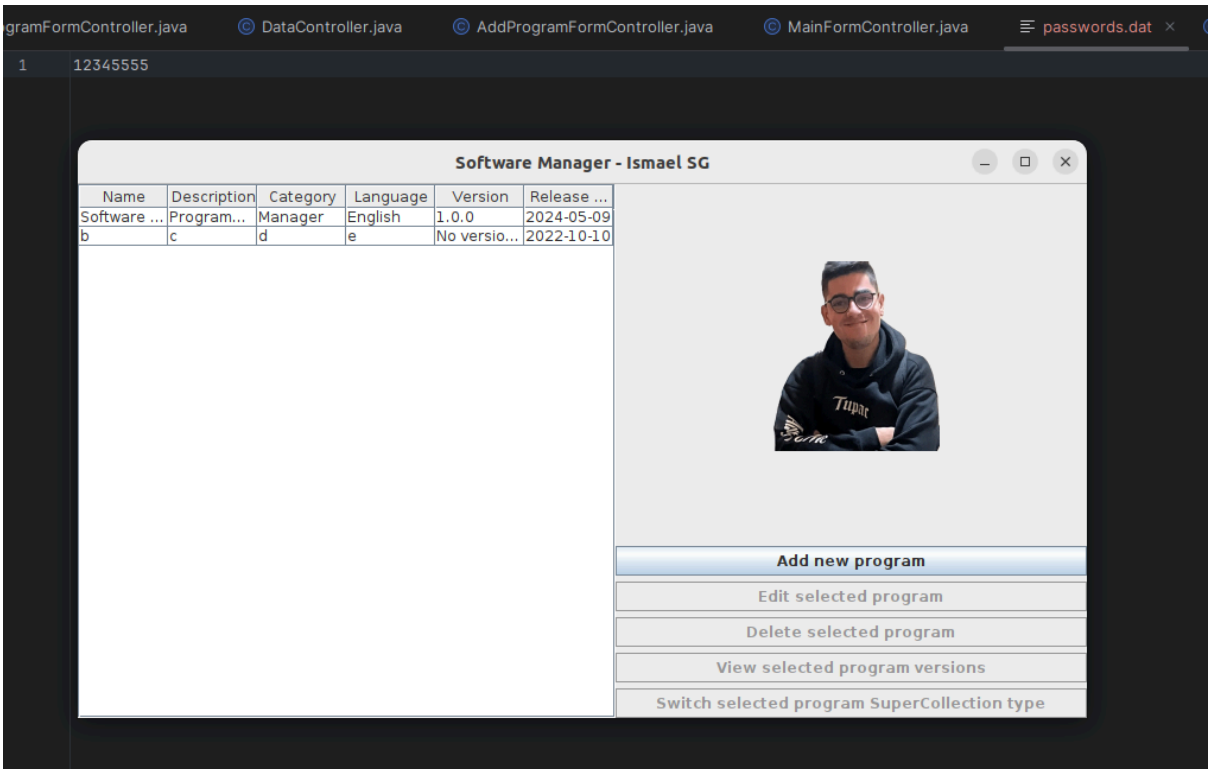
```
this.mainForm.getDeleteButton().addActionListener(e -> {
    if (mainForm.getTable().getSelectedRow() == -1) {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Please select a program to delete!",  title: "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    PasswordPromptForm passwordPromptForm = new PasswordPromptForm();
    passwordPromptForm.setVisible(true);
    passwordPromptForm.addWindowListener((WindowAdapter) windowClosed(e) -> {
            try {
                if (passwordPromptForm.isPasswordSubmittedSuccessfully()) throw new InvalidPasswordException();
                else {
                    int selectedProgramIndex = mainForm.getTable().getSelectedRow();
                    if (PasswordController.checkProgramPassword(selectedProgramIndex, passwordPromptForm.getSubmittedPassword()))
                        throw new InvalidPasswordException();
                    DataController.appData[selectedProgramIndex] = null;
                    DataController.reallocateDataObjects();
                    DataController.saveData();
                    PasswordController.deleteProgramPassword(selectedProgramIndex);
                    DataController.appDataIndex--;
                    AppController.refreshMainForm();
                }
            } catch (Exception ex) {
                ExceptionController.handleException(ex);
            }
    });
});
```

Aquí podem veure com eliminem el programa del mig i darrere el fitxer de contrasenyes.

Abans

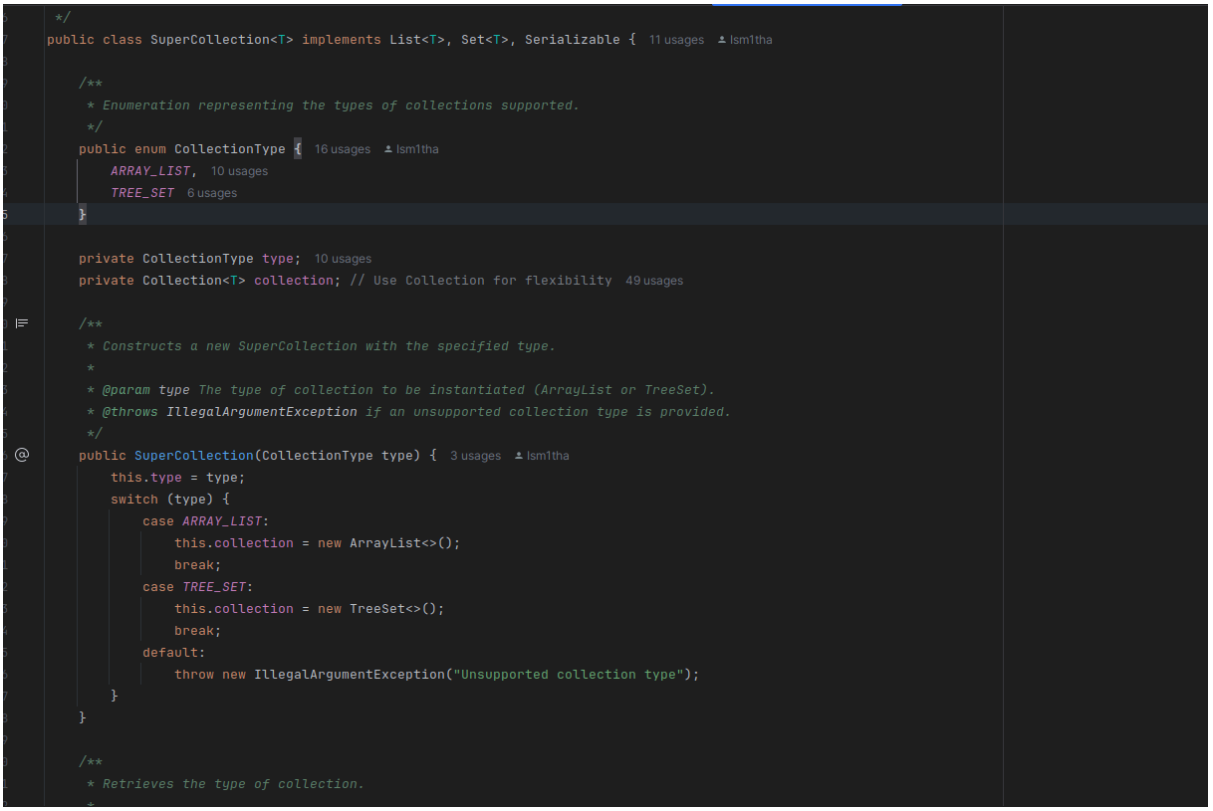


Després



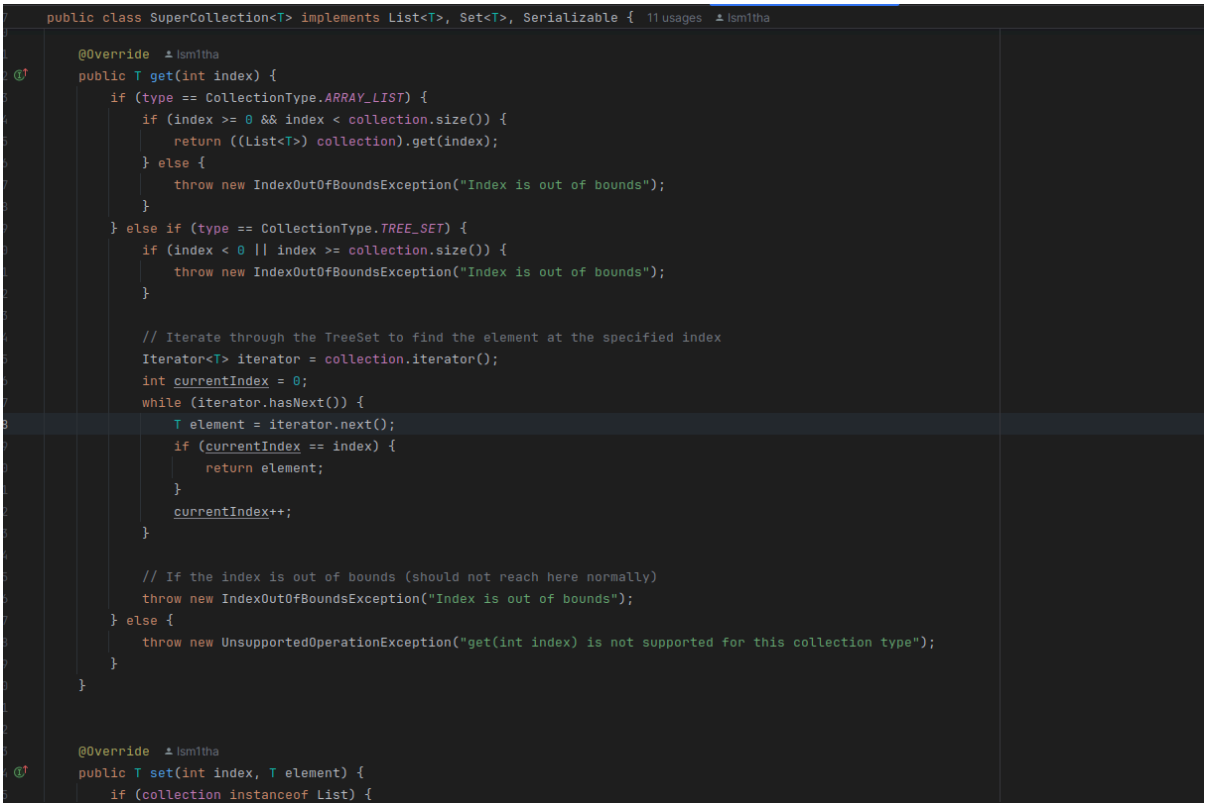
Super Collection

He desenvolupat una classe anomenada SuperCollection que és molt versàtil. Pot comportar-se com una ArrayList o un TreeSet segons la necessitat. Això vol dir que pot emmagatzemar elements d'una manera ordenada (com en un TreeSet) o simplement com una llista (ArrayList) sense cap ordre específic.



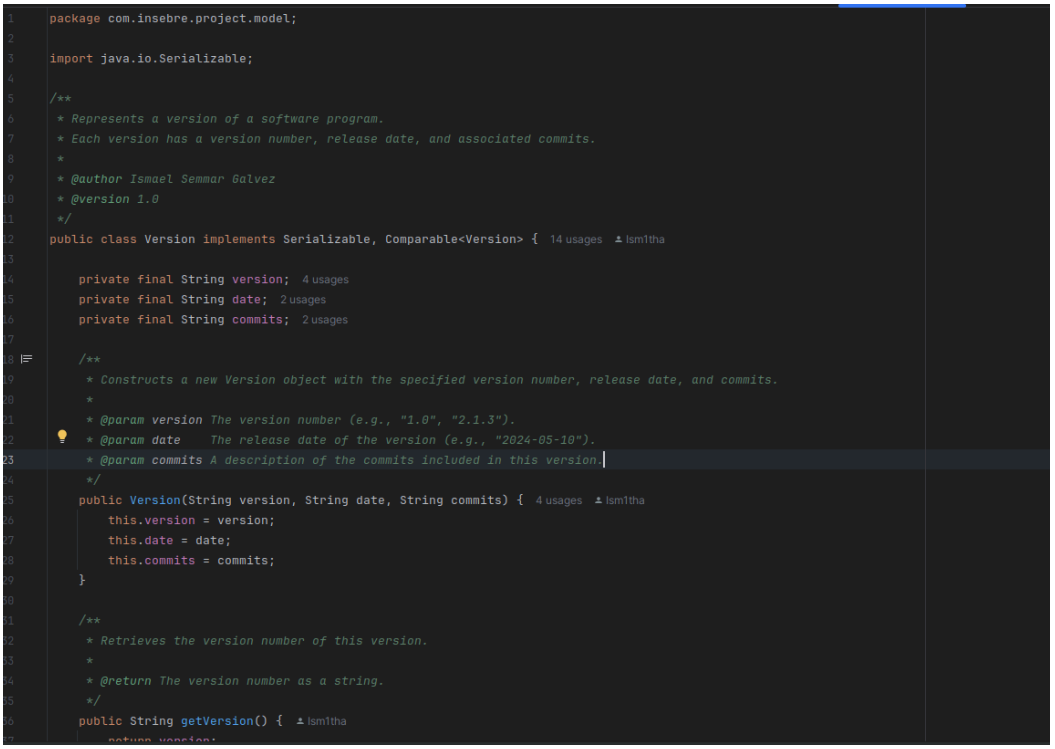
```
 */
public class SuperCollection<T> implements List<T>, Set<T>, Serializable {  11 usages  lsm1tha

    /**
     * Enumeration representing the types of collections supported.
     */
    public enum CollectionType {  16 usages  lsm1tha
        ARRAY_LIST,  10 usages
        TREE_SET  6 usages
    }

    private CollectionType type;  10 usages
    private Collection<T> collection; // Use Collection for flexibility  49 usages

    /**
     * Constructs a new SuperCollection with the specified type.
     *
     * @param type The type of collection to be instantiated (ArrayList or TreeSet).
     * @throws IllegalArgumentException if an unsupported collection type is provided.
     */
    public SuperCollection(CollectionType type) {  3 usages  lsm1tha
        this.type = type;
        switch (type) {
            case ARRAY_LIST:
                this.collection = new ArrayList<>();
                break;
            case TREE_SET:
                this.collection = new TreeSet<>();
                break;
            default:
                throw new IllegalArgumentException("Unsupported collection type");
        }
    }

    /**
     * Retrieves the type of collection.
```

En la classe SuperCollection, he implementat funcions com afegir, eliminar i obtenir elements, i aquestes funcions es comporten de manera diferent segons el tipus de col·lecció que s'hagi seleccionat (ja sigui ArrayList o TreeSet). Per exemple, quan afegim un element a la col·lecció, la funció de afegir té en compte si estem utilitzant una ArrayList o un TreeSet i realitza les accions apropiades per a cada cas.



```java
public class SuperCollection<T> implements List<T>, Set<T>, Serializable {  11 usages  lsm1tha

    @Override  lsm1tha
    public T get(int index) {
        if (type == CollectionType.ARRAY_LIST) {
            if (index >= 0 && index < collection.size()) {
                return ((List<T>) collection).get(index);
            } else {
                throw new IndexOutOfBoundsException("Index is out of bounds");
            }
        } else if (type == CollectionType.TREE_SET) {
            if (index < 0 || index >= collection.size()) {
                throw new IndexOutOfBoundsException("Index is out of bounds");
            }

            // Iterate through the TreeSet to find the element at the specified index
            Iterator<T> iterator = collection.iterator();
            int currentIndex = 0;
            while (iterator.hasNext()) {
                T element = iterator.next();
                if (currentIndex == index) {
                    return element;
                }
                currentIndex++;
            }

            // If the index is out of bounds (should not reach here normally)
            throw new IndexOutOfBoundsException("Index is out of bounds");
        } else {
            throw new UnsupportedOperationException("get(int index) is not supported for this collection type");
        }
    }


    @Override  lsm1tha
    public T set(int index, T element) {
        if (collection instanceof List) {
```

També he creat una altra classe anomenada Version per representar versions de programari. Aquesta classe em permet definir una versió amb el seu número, data de llançament i una descripció dels commits associats. A més, he implementat una funció de comparació per a les versions, de manera que puc comparar-les entre elles basant-me en els números de versió.



```java
package com.insebre.project.model;

import java.io.Serializable;

/**
 * Represents a version of a software program.
 * Each version has a version number, release date, and associated commits.
 *
 * @author Ismael Semmar Galvez
 * @version 1.0
 */
public class Version implements Serializable, Comparable<Version> {  14 usages  lsm1tha

    private final String version;  4 usages
    private final String date;  2 usages
    private final String commits;  2 usages

    /**
     * Constructs a new Version object with the specified version number, release date, and commits.
     *
     * @param version The version number (e.g., "1.0", "2.1.3").
     * @param date    The release date of the version (e.g., "2024-05-10").
     * @param commits A description of the commits included in this version.
     */
    public Version(String version, String date, String commits) {  4 usages  lsm1tha
        this.version = version;
        this.date = date;
        this.commits = commits;
    }

    /**
     * Retrieves the version number of this version.
     *
     * @return The version number as a string.
     */
    public String getVersion() {  lsm1tha
        return version;
```

Un programa conté una Supercollection de versions.

```
*/
public class Program implements Serializable {  22 usages   lsm1tha

    private String name;  3 usages
    private String description;  3 usages
    private String category;  3 usages
    private String language;  3 usages
    private String releaseDate;  3 usages
    private final SuperCollection<Version> versions;  11 usages
```

Les versions poden ser gestionades desde el formulari corresponent.

Gestionar versions

Per gestionar versions simplement fem clic al botó del mainForm “View selected program versions”.

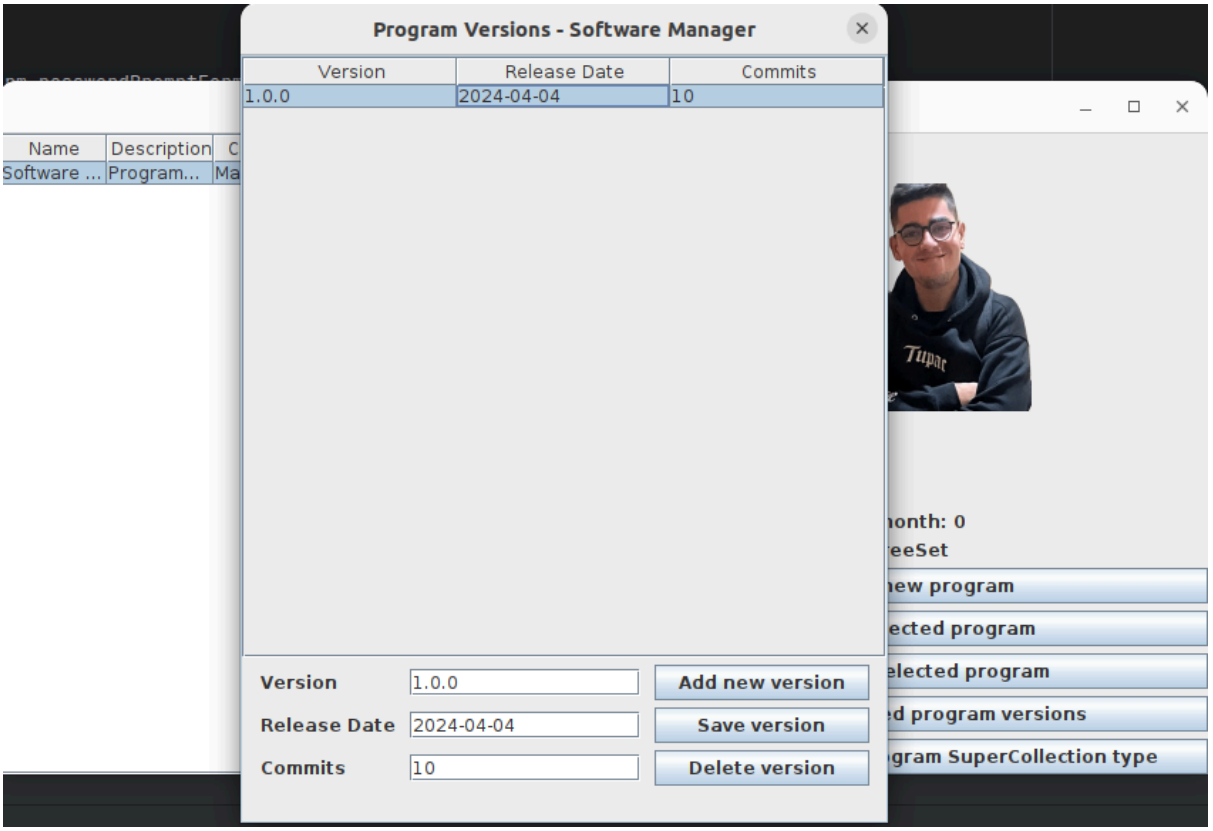
El listener ens obrirà programVersionsForm.

```
// View Selected Program Versions Button ActionListener
this.mainForm.getViewSelectedProgramVersionsButton().addActionListener(e -> {
    if (mainForm.getTable().getSelectedRow() == -1) {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Please select a program to view its versions!",  title: "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    PasswordPromptForm passwordPromptForm = new PasswordPromptForm();
    passwordPromptForm.setVisible(true);
    passwordPromptForm.addWindowListener((WindowAdapter) windowClosed(e) -> {
            try {
                if (passwordPromptForm.isPasswordSubmittedSuccessfully()) throw new InvalidPasswordException();
                else {
                    int selectedProgramIndex = mainForm.getTable().getSelectedRow();
                    if (PasswordController.checkProgramPassword(selectedProgramIndex, passwordPromptForm.getSubmittedPassword()))
                        throw new InvalidPasswordException();
                    if (AppController.showingViewProgramVersionForm) throw new Exception("View Program Versions Form is already open!");
                    ProgramVersionsForm programVersionsForm = new ProgramVersionsForm();
                    ProgramVersionsFormController programVersionsFormController = new ProgramVersionsFormController(programVersionsForm, selectedProgramIndex);
                    programVersionsFormController.setTableData(DataController.getParsedProgramVersions(selectedProgramIndex));
                    programVersionsFormController.show();
                    AppController.showingViewProgramVersionForm = true;
                }
            } catch (Exception ex) {
                ExceptionController.handleException(ex);
            }
    });
});
```

Podem gestionar les versions, el sistema es el mateix que al formulari principal però aquest cop els inputs estan integrats ja a la vista.

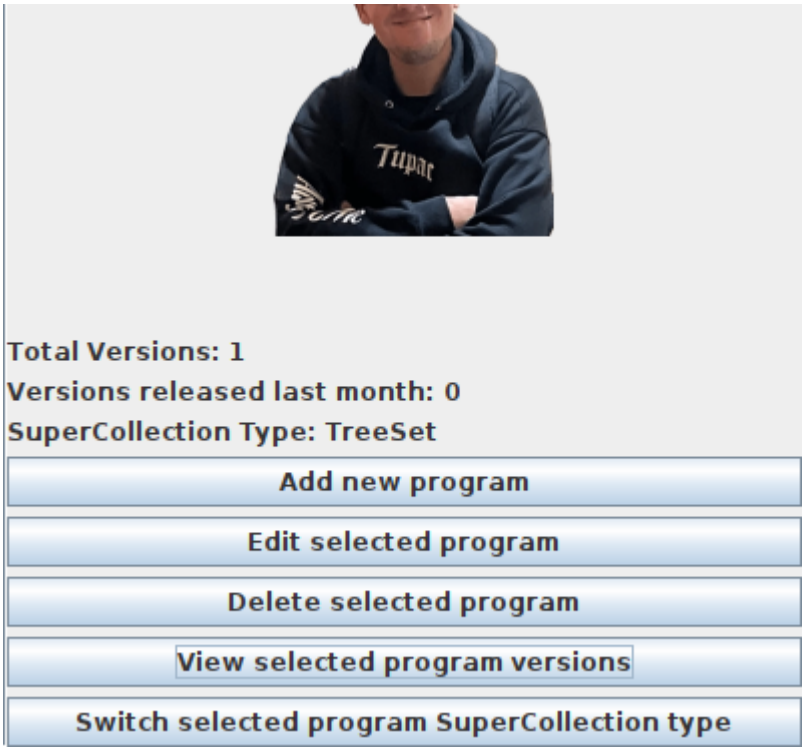
Per realitzar les operacions utilitza els mètodes corresponents al tipo de la supercollection.

Verifica que la nova versió sigui major que la anterior com la data. La data tampoc pot ser futura.



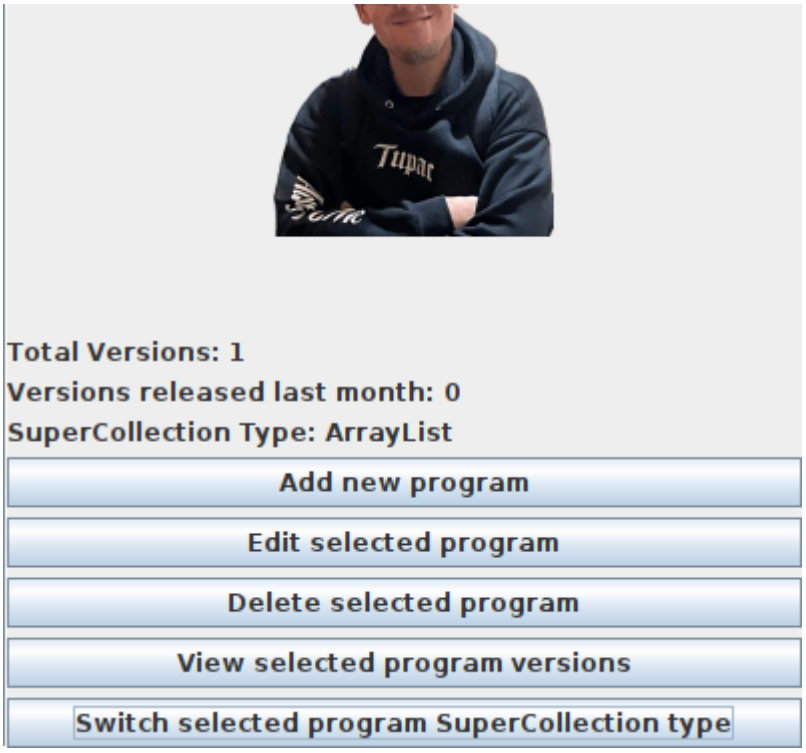
Consultar / Alternar tipo de la Super Collection

En el formulari principal podem veure el tipo de la super collection al fer clic a un programa.



També podem alternar el tipo fent clic a l'últim botó del formulari principal. Al fer clic cridem un mètode de SuperCollection que ens cambiará el tipo de la col·lecció i a part ens traspassarà les dades de un a l'altre.

És important dir que TreeSet elimina els duplicats.



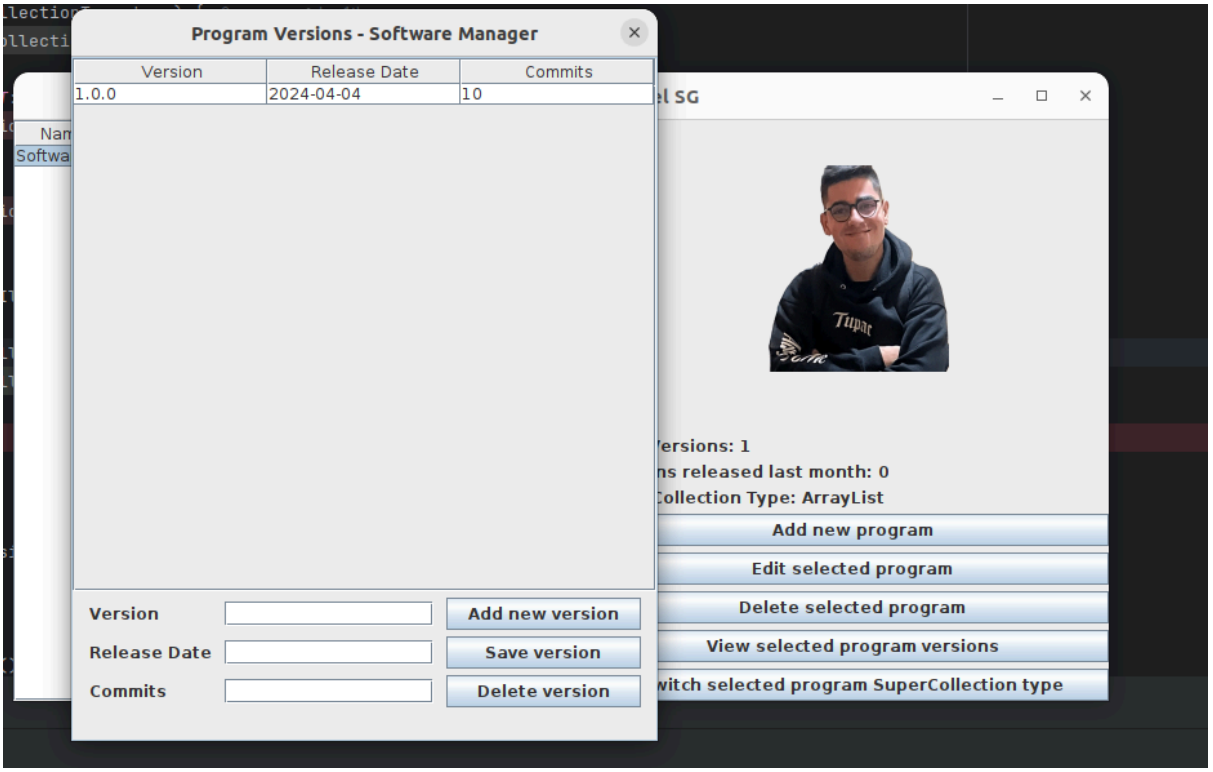

```java
// Switch Selected Program SuperCollection Type Button ActionListener
this.mainForm.getSwitchSelectedProgramSupTypeButton().addActionListener(e -> {
    if (mainForm.getTable().getSelectedRow() == -1) {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Please select a program to switch its SuperCollection type!",  title: "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    PasswordPromptForm passwordPromptForm = new PasswordPromptForm();
    passwordPromptForm.setVisible(true);
    passwordPromptForm.addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosed(WindowEvent e) {
            try {
                if (passwordPromptForm.isPasswordSubmittedSuccessfully()) throw new InvalidPasswordException();
                if (AppController.showingViewProgramVersionForm)
                    throw new Exception("Close the View Program Versions Form before switching the SuperCollection type");
                else {
                    int selectedProgramIndex = mainForm.getTable().getSelectedRow();
                    if (PasswordController.checkProgramPassword(selectedProgramIndex, passwordPromptForm.getSubmittedPassword()))
                        throw new InvalidPasswordException();
                    DataController.switchProgramSuperCollectionType(selectedProgramIndex);
                    DataController.saveData();
                    AppController.refreshSelectedProgramInformation();
                }
            } catch (Exception ex) {
                ExceptionController.handleException(ex);
            }
        }
    });
});
```

Els mètodes implementats a SuperCollection s'ajusten per complir els objectius independentment del tipo i les seves capacitats.

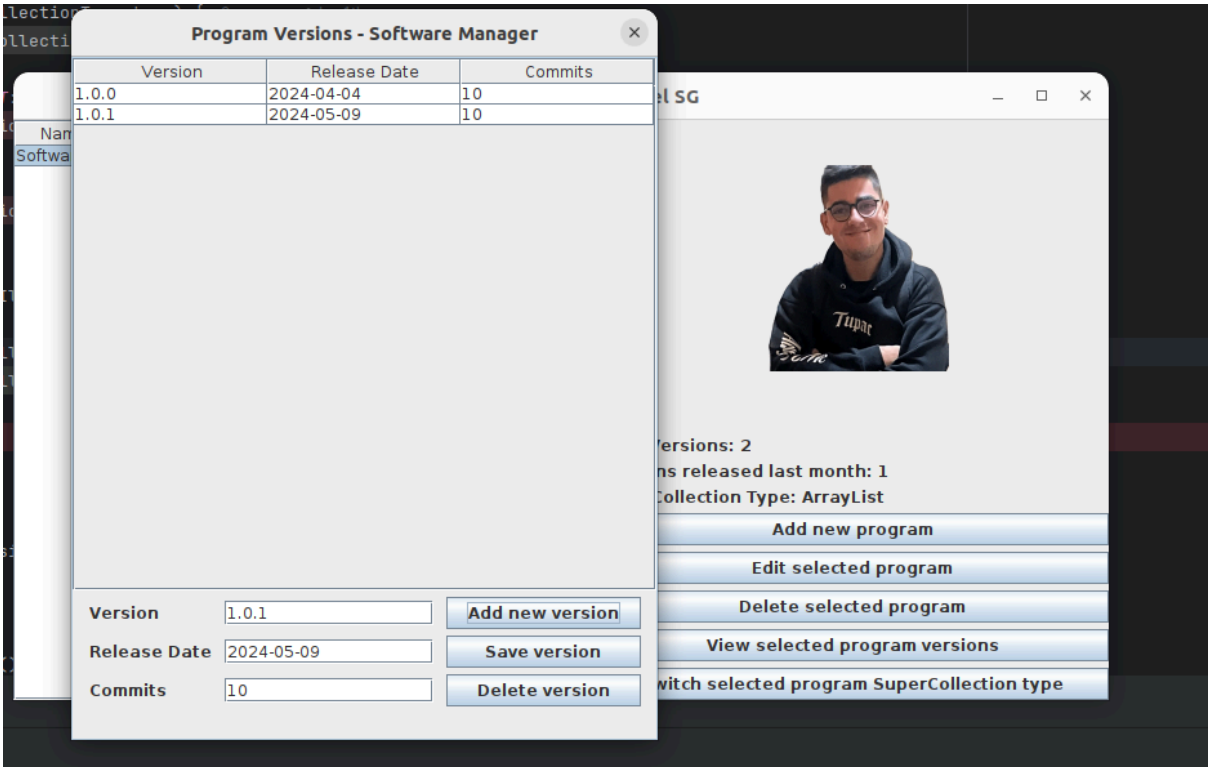
```java
 * Sets the type of collection.
 *
 * @param type The new type of collection (ArrayList or TreeSet).
 * @throws IllegalArgumentException if an unsupported collection type is provided.
 */
public void setType(CollectionType type) {  2 usages   ism1tha
    Collection<T> newCollection;
    switch (type) {
        case ARRAY_LIST:
            newCollection = new ArrayList<>();
            break;
        case TREE_SET:
            newCollection = new TreeSet<>();
            break;
        default:
            throw new IllegalArgumentException("Unsupported collection type");
    }
    newCollection.addAll(collection);
    collection = newCollection;
    this.type = type;
}
```

Streams

Per fer ús dels streams, el que he fet es filtrar les versions del programa seleccionat dintre de la supercollection i he comptat els que son de l'últim mes per mitjà de filter.



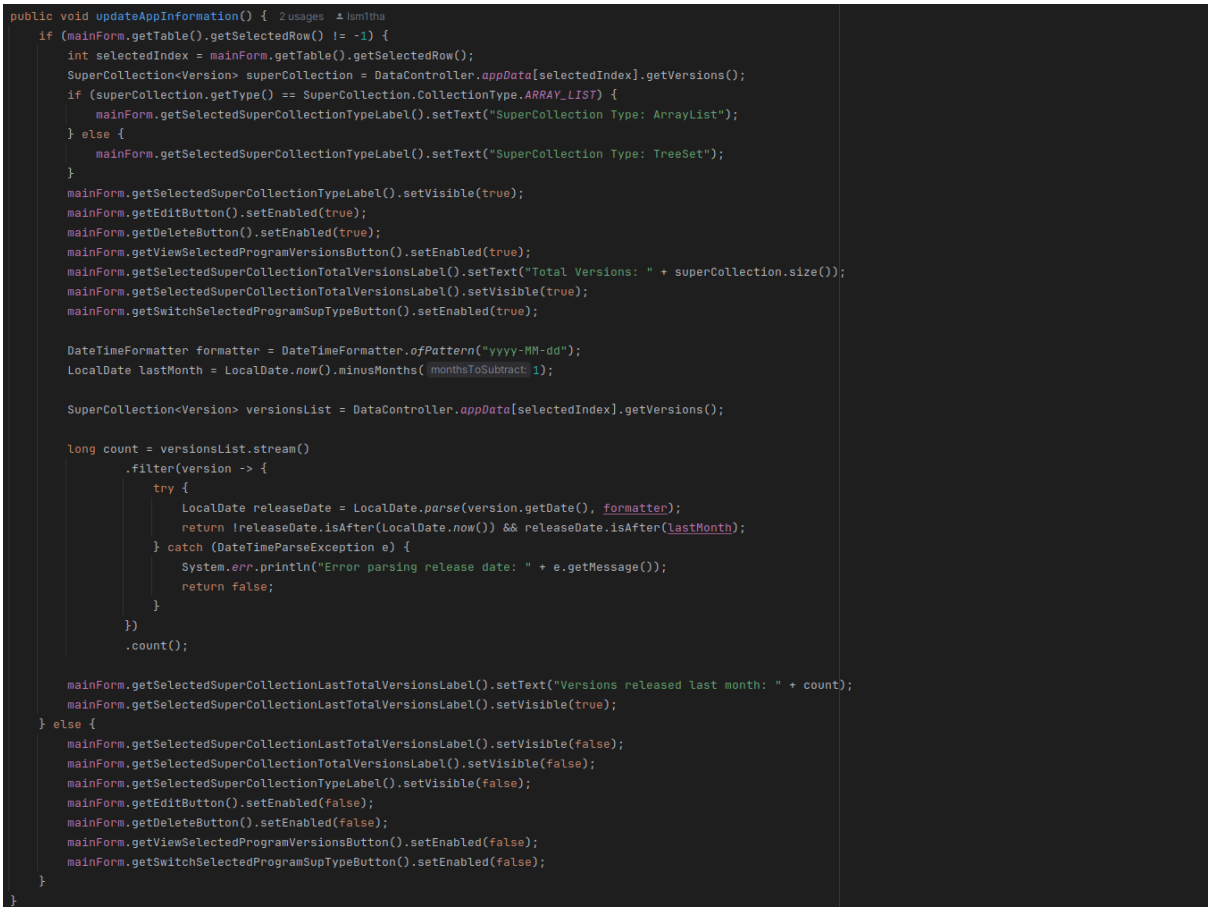
Afegeixo una del nou de Maig.



Versions released last month: 1

```
SuperCollection<Version> versionsList = DataController.appData[selectedIndex].getVersions();

long count = versionsList.stream()
        .filter(version -> {
            try {
                LocalDate releaseDate = LocalDate.parse(version.getDate(), formatter);
                return !releaseDate.isAfter(LocalDate.now()) && releaseDate.isAfter(lastMonth);
            } catch (DateTimeParseException e) {
                System.err.println("Error parsing release date: " + e.getMessage());
                return false;
            }
        })
        .count();
```

Això ho faig en un mètode que tinc al MainFormController per actualitzar la informació del element seleccionat al formulari.



```
public void updateAppInformation() {  2 usages   lsm1tha
    if (mainForm.getTable().getSelectedRow() != -1) {
        int selectedIndex = mainForm.getTable().getSelectedRow();
        SuperCollection<Version> superCollection = DataController.appData[selectedIndex].getVersions();
        if (superCollection.getType() == SuperCollection.CollectionType.ARRAY_LIST) {
            mainForm.getSelectedSuperCollectionTypeLabel().setText("SuperCollection Type: ArrayList");
        } else {
            mainForm.getSelectedSuperCollectionTypeLabel().setText("SuperCollection Type: TreeSet");
        }
        mainForm.getSelectedSuperCollectionTypeLabel().setVisible(true);
        mainForm.getEditButton().setEnabled(true);
        mainForm.getDeleteButton().setEnabled(true);
        mainForm.getViewSelectedProgramVersionsButton().setEnabled(true);
        mainForm.getSelectedSuperCollectionTotalVersionsLabel().setText("Total Versions: " + superCollection.size());
        mainForm.getSelectedSuperCollectionTotalVersionsLabel().setVisible(true);
        mainForm.getSwitchSelectedProgramSupTypeButton().setEnabled(true);

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        LocalDate lastMonth = LocalDate.now().minusMonths( monthsToSubtract: 1);

        SuperCollection<Version> versionsList = DataController.appData[selectedIndex].getVersions();

        long count = versionsList.stream()
                .filter(version -> {
                    try {
                        LocalDate releaseDate = LocalDate.parse(version.getDate(), formatter);
                        return !releaseDate.isAfter(LocalDate.now()) && releaseDate.isAfter(lastMonth);
                    } catch (DateTimeParseException e) {
                        System.err.println("Error parsing release date: " + e.getMessage());
                        return false;
                    }
                })
                .count();

        mainForm.getSelectedSuperCollectionLastTotalVersionsLabel().setText("Versions released last month: " + count);
        mainForm.getSelectedSuperCollectionLastTotalVersionsLabel().setVisible(true);
    } else {
        mainForm.getSelectedSuperCollectionLastTotalVersionsLabel().setVisible(false);
        mainForm.getSelectedSuperCollectionTotalVersionsLabel().setVisible(false);
        mainForm.getSelectedSuperCollectionTypeLabel().setVisible(false);
        mainForm.getEditButton().setEnabled(false);
        mainForm.getDeleteButton().setEnabled(false);
        mainForm.getViewSelectedProgramVersionsButton().setEnabled(false);
        mainForm.getSwitchSelectedProgramSupTypeButton().setEnabled(false);
    }
}
```

Javadocs

He afegit a l'arrel del projecte dintre del repositori de github un directori anomenat javadocs amb tota la documentació del programa per a que puguis consultar qualsevol cosa que hagi faltat explicar.

Conclusió

El projecte no és perfecte, la documentació tampoc, però crec que compleix bastant el que demanes.

Vaig just de temps i no li he pogut dedicar molt més temps en millorar coses com podría ser la interfície gràfica o parts de codi les quals no acabo de estar content del tot de com han quedat.

Igualment crec que el projecte de per sí m'ha servit per entendre el que son genèrics i la resta de coses que hem vist durant aquesta unitat formativa.