OlgaMarcoDAM2 Examen Interfaces

He pensado que entre las cosas que puse en la autoevaluación que quería implementar si tuviese más tiempo, la idea más completa y enriquecedora que considero que puedo hacer, es añadir formato MARKDOWN a mis archivos.

Aclaraciones para que sea más fácil entender el código.

Todos los “TODO” que hay en mi controlador, los he puesto en el examen (para que se te haga más fácil encontrar los cambios)

```
//TODO
```

En la línea 472 del controlador se encuentra el comentario:

```
// --- LÓGICA DE MARKDOWN (Cambios que hice en el examen) ---
```

El cual marca el comienzo del método que implementé nuevo.

Métodos nuevos	Métodos con algún cambio (con un //TODO dentro)

`convertirTextFlowAMarkdown()`
`convertirMarkdownATextFlow()`
`procesarEstilosNegritaCursiva()`
`añadirNodosTexto()`
	`abrirArchivo()`
	`exportarArchivo()`

El método `cerrarInterfaz()` está abajo, pero ya estaba hecho.

Y ese método se hace llamada en el método exportarArchivo();

Cuestión 1: Representación Intermedia

Mi editor de texto incluye un dos botones que tienen que ver con la exportación e importación de archivos. Mi editor funciona con formato readme (Mi implementación en la cuestión dos es la encargada de pasar a formato Markdown, por lo que hablaré de eso en ese apartado)

El método encargado de exportar archivos es `exportarArchivo()`, el cual funciona con el botón de `botonExportar`. (El `//TODO` es lo que añadí en el apartado de implementaciones para que exportarse a readme) Este método exporta usando FileChooser y FileWriter.

```
@FXML
private void exportarArchivo() {
   String contenidoASlavar = convertirTextFlowAMarkdown(); //
Llama al convertidor a Markdown. //TODO

   FileChooser fileChooser = new FileChooser();
   fileChooser.setTitle("Guardar como...");
   fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Archivo de texto", "*.md"));
   File archivo = fileChooser.showSaveDialog(null);
   if (archivo != null) {
       try (BufferedWriter writer = new BufferedWriter(new
FileWriter(archivo))) {
           writer.write(contenidoASlavar);
       } catch (IOException e) { e.printStackTrace(); }
   }

   Thread hilo = new Thread(() -> {
       try{
           for (int i = 1; i <= 5; i++) {
               Thread.sleep(1000);
               final int step = i;
               Platform.runLater(() -> {
                   progressLabel.setProgress(step / 5.0);
                   progressLabel.setText("Guardando... " + (step *
20) + "%");
               });
           }
       }catch(InterruptedException e){
           e.printStackTrace();
       }
   });

   hilo.start();
}
```

El método encargado de exportar archivos es `abrirArchivo()`, el cual funciona con el botón de `botonAbrir`. Este método exporta usando FileChooser y BufferedReader.

```
@FXML
```

```
private void abrirArchivo() {
   FileChooser fileChooser = new FileChooser();
   fileChooser.setTitle("Abrir archivo de texto");
   fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Archivo de texto", "*.md"));
   File archivo = fileChooser.showOpenDialog(null);
   if (archivo != null) {
       try (BufferedReader reader = new BufferedReader(new
FileReader(archivo))) {
           String contenido =
reader.lines().collect(Collectors.joining("\n"));

           textArea.setText(contenido);
           convertirMarkdownATextFlow(contenido); //TODO
           isFirstChange = false;
           isGlobalBold = false; isGlobalItalic = false;
isGlobalUnderline = false;
           historial();
       } catch (IOException e) { e.printStackTrace(); }
   }
}
```

Cuestión 2: Implementaciones

1.Ejemplo de la funcionalidad

Si exporto un archivo .md en mi editor….

<u>EJEMPLO TEXTO QUE HE IMPORTADO</u> (Texto con negrita, cursiva y subrayado)

Quiero que a la hora de abrirlo en un IDE. Se vea en formato markdown, en este caso sería…

<u>***EJEMPLO TEXTO QUE HE IMPORTADO***</u>

Y que al volver a importarlo con el botón de abrir, se vea de nuevo como al principio:

<u>EJEMPLO TEXTO QUE HE IMPORTADO</u> (Texto con negrita, cursiva y subrayado)

2.¿Cómo pienso implementarlo?

1. Debo crear un método que cuando ve que los botones negrita, cursiva y subrayado estan activos a la hora de exportar el archivo, respectivamente, los convierta en formato markdown. ¿Cómo hago eso? Si el texto está en negrita y cursiva, que ponga ***, si está en negrita solo, que ponga **, si está en cursiva solo * y si esta subrayado que añada <u> al principio y </u> al final. Creo que usare un StringBuilder para organizarme al almacenar cosas y este método, lo inicializaré en el método de exportarArchivos ya creado.

2. Luego crearé otro método que cuando abro un archivo en formato readme, que tiene ***, *, **, o <u> al principio y </u> al final. Pase de nuevo a negrita, cursiva, subrayado o mezcladas (Dependiendo de lo que tenga)

3.Implementación.

1.Método `convertirTextFlowAMarkdown()`

Este método usa un stringbuilder para añadir el texto y usa un nodo para iterar el texto del textFlow sobre cada nodo. Gracias a un String que he llamado texto, paso el formato del texto (negrita itálica y subrayado) a el mismo formato en Markdown (Ósea *** ** * y </u>)

```
private String convertirTextFlowAMarkdown() {
   StringBuilder markdownBuilder = new StringBuilder(); //
StringBuilder para añadir el texto
   for (Node node : textFlow.getChildren()) { // Itera sobre
cada nodo (fragmento de texto) del TextFlow
       if (node instanceof Text) { // Si el nodo es de tipo
texto
           Text textNode = (Text) node; // lo convierte a la
clase Text
           String texto = textNode.getText(); // Obtiene el
contenido de texto plano del nodo

           // Evita añadir delimitadores Markdown a texto
vacío o que solo contiene espacios
           if (texto.trim().isEmpty()) { // Si el texto está
vacío o es solo espacio
               markdownBuilder.append(texto); // lo añade tal
cual
               continue; // y pasa al siguiente nodo
```

```
            }

            String estilo =
textNode.getFont().getStyle().toLowerCase(); // Obtiene el
estilo de la fuente en minúsculas
            boolean esNegrita = estilo.contains("bold"); //
Comprueba si el estilo contiene negrita
            boolean esCursiva = estilo.contains("italic"); //
Comprueba si el estilo contiene cursiva
            boolean esSubrayado = textNode.isUnderline(); //
Comprueba si el texto está subrayado

            String textoReadme = texto; // Inicializa el texto
Markdown con el texto plano

            if (esNegrita && esCursiva) { // Si es negrita y
cursiva
                textoReadme = "***" + textoReadme + "***"; //
lo pone con tres asteriscos
            } else if (esNegrita) { // Si es solo negrita,
                textoReadme = "**" + textoReadme + "**"; // lo
pone con dos asteriscos
            } else if (esCursiva) { // Si es solo cursiva,
                textoReadme = "*" + textoReadme + "*"; // lo
pone con un asterisco
            }

            if (esSubrayado) {
                textoReadme = "<u>" + textoReadme + "</u>"; //
lo envuelve con etiquetas HTML de subrayado
            }
            markdownBuilder.append(textoReadme); // Añade el
texto con formato Markdown al resultado final
        }
    }
    return markdownBuilder.toString(); // Devuelve el readme.
}
```

Luego en el método exportarArchivo, lo llamo para hacer que funcione con el string “Contenido a salvar”.

```
private void exportarArchivo() {
    String contenidoASlavar = convertirTextFlowAMarkdown();
// Llama al convertidor a Markdown. //TODO
```

2.Método `convertirMarkdownATextFlow()`

Este método hace lo contrario. Coje el readme y lo vuelve a pasar al formato en el textflow a la hora de abrir el archivo. Primero limpio el textFlow para que no se rallen. Ahí con ese pattern hago que encuentre los patrones del subrayado en el formato Markdown y los cambia a subrayado. Finalmente uso el metodo que hice para la italica y negrita (lo hice en otro método porque aquí todo junto no funcionaba) para que si tiene negrita e italica no se confunda con el subrayado.

```
private void convertirMarkdownATextFlow(String markdown) {
    // Limpia el contenido actual del TextFlow para empezar de cero
(Porque si hay texto ya, se mezcla con el texto que ya hay)
    textFlow.getChildren().clear();

    // El procesamiento se hace en dos fases:
    // 1. Se identifican los bloques de texto subrayados
(<u>...</u>).
    // 2. Dentro de cada bloque (subrayado o no), se procesan los
estilos de negrita y cursiva.

    // Patrón para encontrar texto envuelto en etiquetas <u></u>.
    // Pattern.DOTALL permite que '.' incluya saltos de línea, para
que el subrayado pueda abarcar múltiples líneas.
    Pattern patronSubrayado = Pattern.compile("<u>(.*?)</u>",
Pattern.DOTALL);
    Matcher matcherSubrayado = patronSubrayado.matcher(markdown);

    int ultimaPosicion = 0; // Para rastrear qué partes del texto
ya se han procesado.

    // Itera sobre todas las coincidencias de texto subrayado.
    while (matcherSubrayado.find()) {
        // Procesa el texto que está ANTES de la sección subrayada.
        // Este texto, por definición, no está subrayado.
        if (matcherSubrayado.start() > ultimaPosicion) {
            String textoPrevio = markdown.substring(ultimaPosicion,
matcherSubrayado.start());
            procesarEstilosNegritaCursiva(textoPrevio, false); //
false porque no está subrayado.
        }

        // Procesa el texto que está DENTRO de las etiquetas
<u></u>.
```

```
            String textoSubrayado = matcherSubrayado.group(1); //
group(1) es el contenido entre las etiquetas.
            procesarEstilosNegritaCursiva(textoSubrayado, true); //
true porque sí está subrayado.

            // Actualiza la última posición procesada.
            ultimaPosicion = matcherSubrayado.end();
        }

        // Procesa el texto restante que queda DESPUÉS de la última
sección subrayada.
        // Este texto tampoco está subrayado.
        if (ultimaPosicion < markdown.length()) {
            String textoRestante = markdown.substring(ultimaPosicion);
            procesarEstilosNegritaCursiva(textoRestante, false);
        }
    }
```

Luego en el método abrirArchivo, lo llamo para hacer que al abrir un archivo, se ejecute el método.

```
@FXML
private void abrirArchivo() {
convertirMarkdownATextFlow(contenido); //TODO:
```

3.Método `procesarEstilosNegritaCursiva()`

Este método identifica los de *** ** y * de Negrita o cursiva. Lo hago con un Matcher para mirar la posición. (lo intenté con el reader pero daba error, así que me pasé a esto) y con booleanos hago que se active la negrita e italica en el textflow. Este método da uso dentro de convertirMarkdownATextFlow() y como este se conectaba a abrirArchivo , este método se procesa al abrir un archivo.

```
private void procesarEstilosNegritaCursiva(String texto,
boolean esSubrayado) {
    // Patrón para identificar negrita-cursiva (***), negrita
(**), y cursiva (*).
    // El orden es importante para que no confunda *** con ** o
*.
    final Pattern patronEstilos =
Pattern.compile("(\\*\\*\\*(.*?)\\*\\*\\*)|(\\*\\*(.*?)\\*\\*)
|(\\*([^*]+?)\\*)");
    Matcher matcherEstilos = patronEstilos.matcher(texto);
```

```
   int ultimaPosicion = 0; // Rastrea la posición en el
fragmento de texto actual.

   // Itera sobre todas las coincidencias de estilos.
   while (matcherEstilos.find()) {
       // Añade el texto que se encuentra antes del siguiente
estilo.
       if (matcherEstilos.start() > ultimaPosicion) {
           String textoSinEstilo =
texto.substring(ultimaPosicion, matcherEstilos.start());
           añadirNodosTexto(textoSinEstilo, false, false,
esSubrayado);
       }

       String contenido;
       boolean esNegrita = false;
       boolean esCursiva = false;

       if (matcherEstilos.group(1) != null) { // Coincidencia
de ***...***
           contenido = matcherEstilos.group(2);
           esNegrita = true;
           esCursiva = true;
       } else if (matcherEstilos.group(3) != null) { //
Coincidencia de **...**
           contenido = matcherEstilos.group(4);
           esNegrita = true;
       } else { // Coincidencia de *...*
           contenido = matcherEstilos.group(6);
           esCursiva = true;
       }

       // Añade el texto con los estilos correspondientes.
       añadirNodosTexto(contenido, esNegrita, esCursiva,
esSubrayado);
       ultimaPosicion = matcherEstilos.end(); // Actualiza la
posición.
   }

   // Añade el texto restante que no tenía ningún estilo de
negrita/cursiva.
   if (ultimaPosicion < texto.length()) {
       String textoRestante = texto.substring(ultimaPosicion);
```

```
        añadirNodosTexto(textoRestante, false, false,
esSubrayado);
    }
}
```

4.Método `añadirNodosTexto()`

Este método es para añadir el texto editado con negrita cursiva y subrayado con esos formatos al textflow con FontWeight y FontPosture.

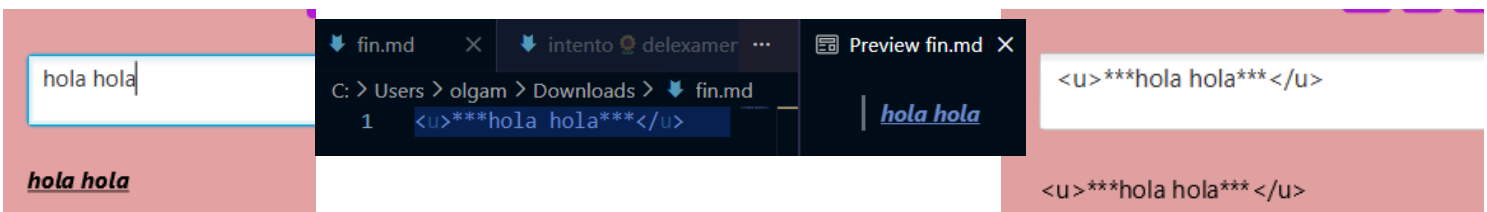
```
private void añadirNodosTexto(String content, boolean negrita,
boolean cursiva, boolean subrayado) {
    if (content.isEmpty()) return;

    String[] lines = content.split("\\n", -1);
    for (int i = 0; i < lines.length; i++) {
        if (!lines[i].isEmpty()) {
            Text textNode = new Text(lines[i]);
            FontWeight weight = negrita ? FontWeight.BOLD :
FontWeight.NORMAL;
            FontPosture posture = cursiva ? FontPosture.ITALIC :
FontPosture.REGULAR;
            textNode.setFont(Font.font("System", weight, posture,
12));
            textNode.setUnderline(subrayado);
            textFlow.getChildren().add(textNode);
        }

        if (i < lines.length - 1) {
            textFlow.getChildren().add(new Text("\n"));
        }
    }
}
```

4.Capturas de como funcionaba el editor conforme

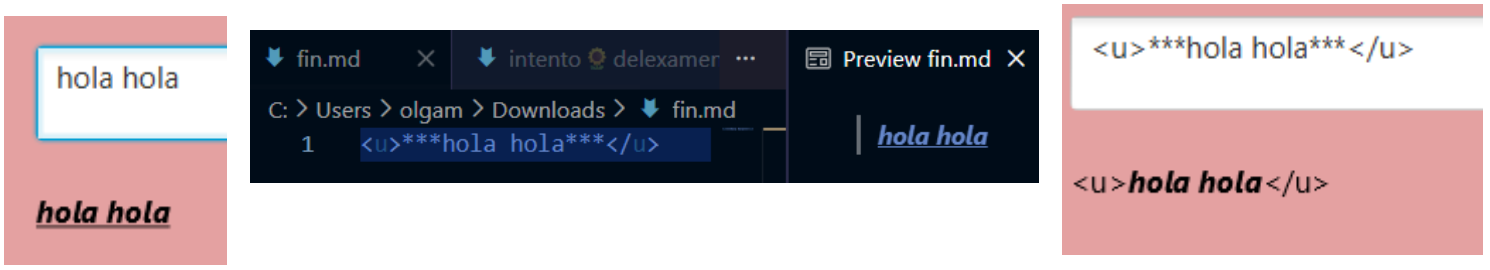
Capturas de como funcionaba el editor conforme iba añadiendo cosas.

1.Después de añadir el primer método creado llamado `convertirTextFlowAMarkdown()`, el editor, al exportar un archivo como el de la primera imagen, lo guarda en formato markdown de manera que VS code se ve como en la segunda imagen. Sin embargo, en la tercera captura podemos ver como el formato no se vuelve a restaurar a la hora de abrirlo en el editor de nuevo.

¡Ya tenemos tarea por hacer!🐱



2.1 Después de añadir el método `convertirTextFlowAMarkdown()`, implementé el método `convertirMarkdownATextFlow()`, que hace lo contrario, básicamente. Pilla el texto del markdown y lo vuelve a pasar al formato que había en el textflow. Se que no lo consigo a medias, pero en el método lo he intentado implementar y así queda al abrirlo (podría ser peor) :(



2.2 Al final lo solucioné :D. Lo hice creando dos nuevas clases que he explicado arriba. `añadirNodosTexto y procesarEstilosNegritaCursiva.` Pero resumidamente, al cambiar negrita y italica de metodo, se hace el subrayado. Y añadí que se quitasen los <u> y </u>