2. LinkTracker

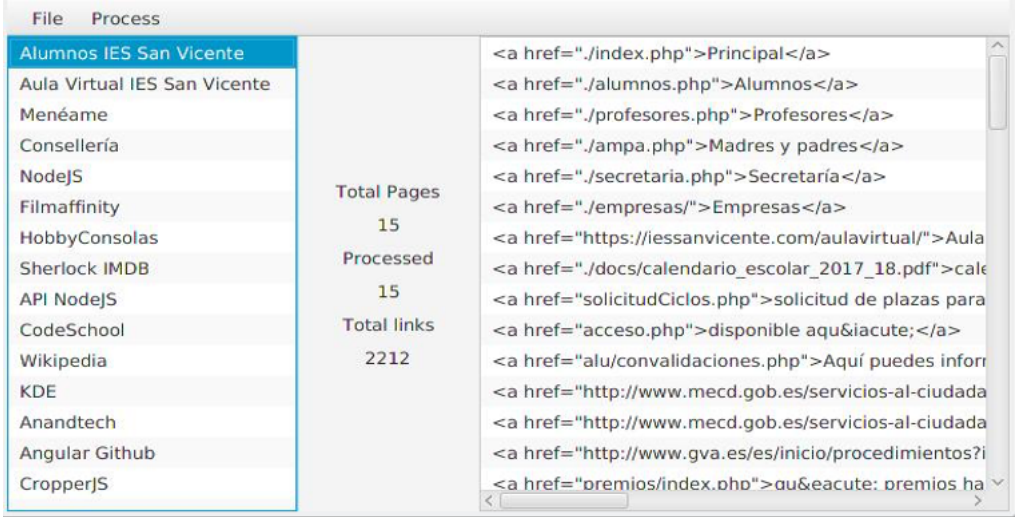
Unidad 2. Ejercicio final

Programación de Servicios y Procesos

Tabla de contenido

1. Introducción y primeros pasos

En este ejercicio final, se pedirá que implementes una aplicación JavaFX capaz de manejar múltiples hilos para acceder a páginas web remotas y recopilar todos los enlaces que se encuentren en ellas. La apariencia de la aplicación será más o menos así:



1.1. Configurar el proyecto

Se creará un proyecto JavaFX (aplicación FXML), llamado ***LinkTracker***. Una vez creado el proyecto, dentro de la sección SRC, crea los siguientes paquetes y sub-paquetes:

Una vez creado el proyecto, en paquete principal de la sección src>main>java crea la siguiente estructura:

- El paquete principal *com.example.linktracker*, contendrá la clase principal y los controladores de JavaFX.
- El paquete *com.example.linktracker.**model***, para almacenar el modelo (clase *WebPage*, se explica más adelante).
- El paquete *com.example.linktracker.**utils***, para almacenar algunas clases útiles.

> **NOTA**: sustituye *example* por tu nombre y primer apellido.

2. Estructura de clase

Además de la aplicación principal JavaFX con el archivo FXML y el controlador (se verán en la siguiente sección), se necesitarán algunas clases adicionales para almacenar la información sobre las páginas web.

2.1. La clase *WebPage*

Añade una nueva clase dentro del paquete ***linktracker.model*** llamada ***WebPage***. Esta clase tendrá los siguientes atributos:

- El nombre de la página web (un *string*, como “IG Formación”).
- La URL de la web (otro *string*).
- La lista de enlaces recopilados de esta página (una lista de *strings*).

Además, se añadirá un constructor con el nombre de la página y la URL (la lista de enlaces se completará más adelante, después de crear el objeto). Finalmente, añade los *getters* y *setters* para cada atributo.

2.2. Clase *FileUtils*

Para que la lista de páginas web se procese, se creará una clase llamada ***FileUtils*** en el paquete ***linktracker.utils***. Esta clase tendrá un método estático llamado ***loadPages()***, que recibirá un *Path* como parámetro, y devolverá una lista de objetos *WebPage*:

```
static List<WebPage> loadPages(Path file)
```

2.2.1. Estructura de los archivos de páginas web

El archivo pasado al método *loadPages()* como parámetro será un archivo de texto con la siguiente estructura:

```
page_name; url
```

Por ejemplo:

```
Aula Virtual IG Formación;http://igformacion.online/
Menéame;https://www.meneame.net
…
```

Donde los atributos están separados por ‘;’.

2.3. Clase *LinkReader*

Se proporcionará esta clase. Contiene un método estático público llamado ***getLinks()***, que recibe una URL como parámetro. Internamente, esta clase se conecta a esta URL, analiza su contenido y obtiene una lista de todos los enlaces (etiquetas “a”) contenidos en esta página. Por tanto, no tienes que preocuparte (por ahora) sobre cómo obtener los enlaces desde una URL remota. Simplemente añade esta clase en tu paquete ***linktracker.utils***.

2.4. Otras clases útiles

Aunque no es obligatorio, puede ser útil añadir algunas otras clases. Por ejemplo, una clase llamada *MessageUtils* (en el paquete *utils*) para mostrar diferentes mensajes aviso (*Alert*). Podría contener los siguientes métodos estáticos:

- ***static void showError(String message)*** para mostrar mensajes de error.
- ***static void showMessage(String message)*** para mostrar mensajes de información.
- …

Puedes añadir tantas clases como necesites dentro del paquete *utils*.

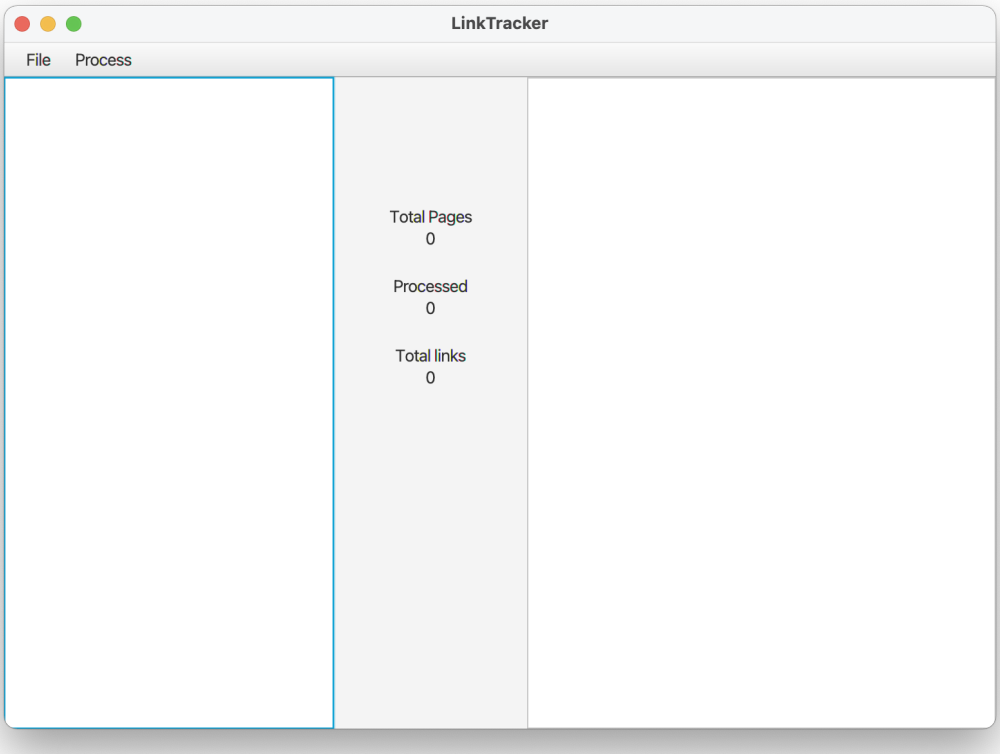
3. La aplicación JavaFX

Crea las clases de aplicación JavaFX. Sigue estos pasos:

1. La aplicación principal de JavaFX se llamará ***LinkTracker*** dentro del paquete ***linktracker***.
2. Habrá un archivo FXML llamado ***FXMLMainView.fxml*** con su controlador asociado (*FXMLMainViewController.java*). El archivos FXML se colocará en el paquete *com.example.linktracker* de la carpeta *resources*, el controlador se ubicará en el paquete principal de la carpeta *java*.
3. Asegúrate de que la clase principal *LinkTracker* carga el contenido del archivo FXML.

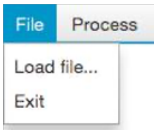
3.1. Diseñando la vista principal

Ahora, haciendo uso de *SceneBuilder* para diseñar la escena principal, intenta obtener una apariencia similar a la siguiente:

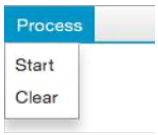


Puedes utilizar un *layout BorderPane* (por ejemplo) para organizar todos los elementos:

- Una barra de menús en la parte superior de la escena con los menús *File* y *Process*.
  - Dentro de *File* se añadirán dos elementos de menús, uno llamado *"Load file…"* y otro llamado *"Exit"*.



- Dentro de Process se añadirán dos elementos de menús, uno será *"Start"* y el otro *"Clear"*, respectivamente.



- Añade un *ListView* en la parte izquierda para cargar las páginas web y otro en el lado derecho para cargar los enlaces que contenga la página web.
- Añade también una etiquetas en el centro para mostrar información sobre el proceso:
  - *Total pages* → Páginas cargadas desde el archivo.
  - *Processed* → Cuantas páginas se han procesado (hilos terminados).
  - *Total links* → Cuantos enlaces se han encontrado entre todas las páginas.

Recuerda establecer un *fx:id* para cada elemento al que pueda ser necesario acceder desde el controlador.

3.2. ¿Cómo debería funcionar?

Al iniciar, la aplicación deberá mostrarse como puede verse arriba, y cuando se elija un elemento de menú, deberá responder al evento.

3.2.1. Cargar un archivo

Si se selecciona el elemento de menú *"Load file…"* del menú *File*, deberá aparecer un cuadro de dialogo *FileChooser*, dónde se podrá seleccionar un archivo de texto con el formato especificado en la sección 2.2.1. de este enunciado. Tan pronto como se elija el archivo, se deberá llamar al método *FileUtils.loadPages()* para cargar una lista de elementos *WebPage* desde el archivo. Después, deberá mostrar un mensaje mediante un diálogo de información que indique cuántas páginas web se han cargado desde el archivo y actualizar la etiqueta correspondiente en la vista principal:

3.2.2. Iniciando el proceso

Si se hace clic en el elemento *"Start"* del menú *Process*, se lanzarán tantos hilos como páginas web cargadas desde el archivo. Si aún no se ha cargado ninguna página web, debe mostrarse un mensaje de error:

Debe haber un *Executor* para manejar todos los hilos. Utiliza un *ThreadPoolExecutor* para poder verificar en cualquier momento cuántas tareas (hilos) han finalizado:

```
(ThreadPoolExecutor) Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
```

Cada tarea debe ser un *Callable* y recibirá un objeto *WebPage* para procesarlo, devolviéndolo de nuevo cuando se hayan obtenido las URL. Utiliza *executor.submit(Callable)* para lanzar cada hilo (*invokeAll* congelaría el hilo principal) y añadir el objeto *Future<WebPage>* devuelto a la lista de enlaces.

Dentro de la tarea, obtén los enlaces de la página web actual (consulta la clase *LinkReader*), actualiza la variable ***"total links"*** añadiendo el número de enlaces encontrados (utiliza una variable atómica) y devuelve la página web con la lista de enlaces obtenida.

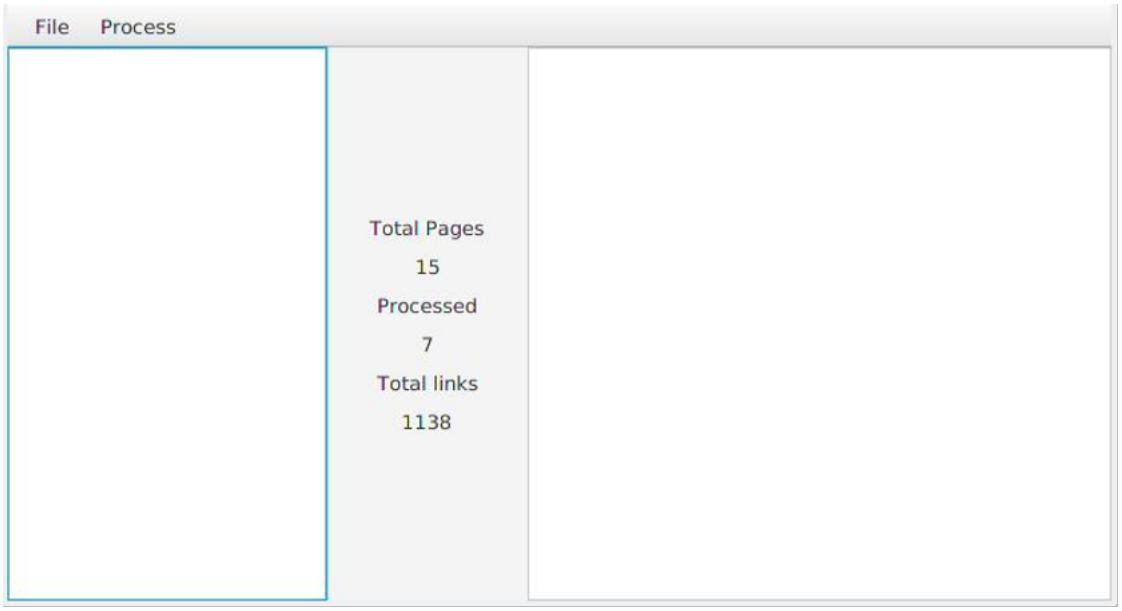
3.2.3. Controlando el progreso

Crea también un JavaFX *ScheduledService* que se ejecutará cada 100 ms. Este servicio retornará si el ejecutor ha terminado.

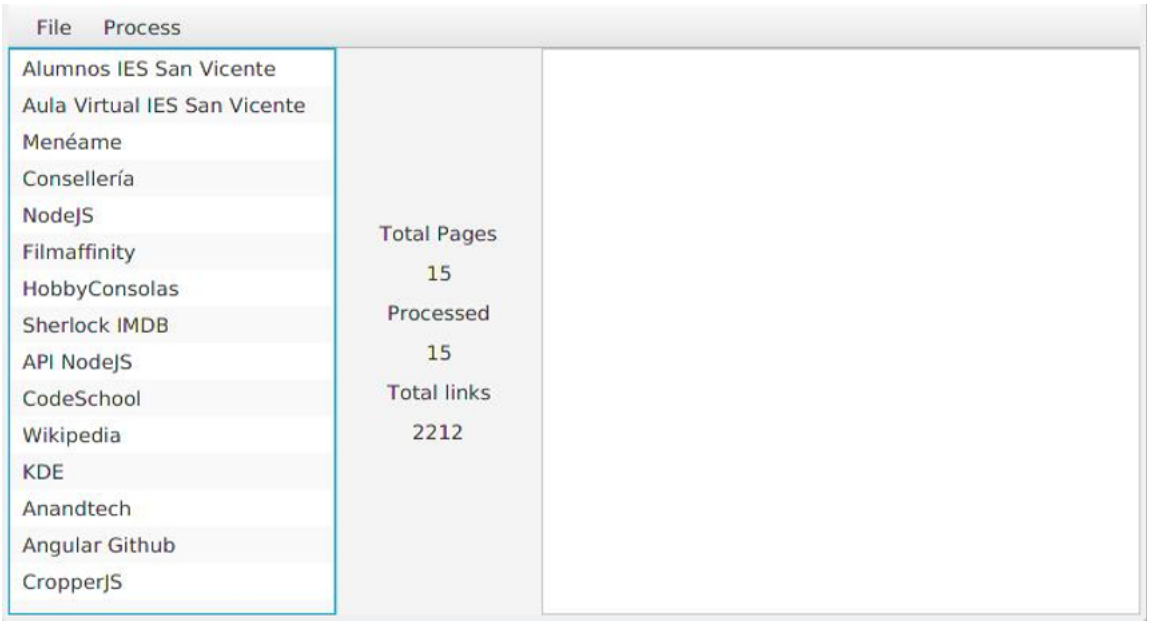
Cada vez que se ejecute el servicio, actualizará las etiquetas *"Processed"* y *"Total links"* (solo los números). Además, cuando se detecte que todas las tareas han finalizado, obtendrá los objetos *WebPage* de *Futures* y se añadirán al *ListView*.

Importante: No utilices *Platform.runLater*. En su lugar, utiliza el evento ***onSucceeded*** de *Service* para actualizar las variables en el hilo principal.

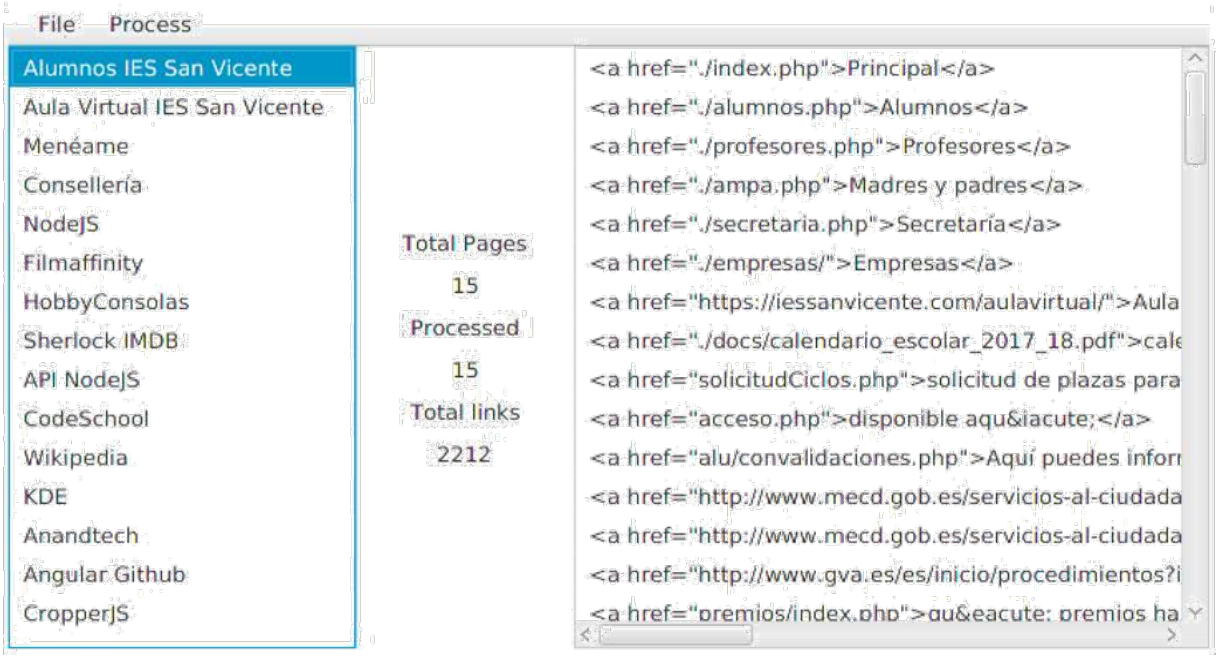
Ejemplo de la aplicación del procesamiento de páginas web:



Ejemplo de resultado final:



Cuando se haga clic en cualquier página web de la lista de la izquierda, entonces se mostrará su lista de enlaces en la lista de la derecha.



3.2.4. Limpiando el proceso

Si se selecciona la opción *"Clear"* del menú de *Process*, todas las listas (lista de la izquierda, lista de la derecha y lista de la página web interna, si corresponde) y los valores de las etiquetas deben borrarse, para que se pueda comenzar un nuevo proceso desde *Start.*

3.2.5. Salir de la aplicación

Finalmente, si se selecciona la opción el *“Exit”* del menú *File*, deberá cerrarse la aplicación.

4. Mejoras opcionales

- Utilizar una *ConcurrentLinkedDeque* para almacenar las páginas web que hayan terminado.
  - Cuando una página termine de cargar enlaces, añadir el objeto WebPage a esta cola concurrente.
  - Tras cada ejecución de *ScheduledService*, añadir los elementos de esta colección concurrente al *ListView* de la izquierda. De esta forma, cada vez que una página termine de cargarse, se verá aparecer en la lista.
- Usar *CompletableFutures* en lugar de *Callable* + *Executor* para las tareas.
  - Como no hay *Executor*, para saber cuántas tareas han finalizado, verica el tamaño de la *ConcurrentLinkedDeque* donde están las páginas terminadas.
  - Cuando una *CompletableFuture* termine de cargar los enlaces de una página web, devolverá la página web y encadenará otra tarea (use ***thenAccept*** por ejemplo), actualiza el número total de enlaces y añade la *WebPage* a la colección *ConcurrentLinkedDeque*. (Primera tarea → obtener enlaces, Segunda tarea → Actualizar el número total y añadir el objeto *WebPage*).

5. Reglas de evaluación

5.1. Parte obligatoria

Para obtener la nota final, se aplicarán los siguientes criterios:

- Estructura de clases (paquetes model y utils), con las clases Flight y FileUtils con el código correspondiente, y todas las demás clases útiles adicionales que puedas necesitar: 1 punto.
- Diseño de la aplicación JavaFX, similar al mostrado en las figuras anteriores: 0,5 puntos.
- Cargado de páginas web desde el archivo de texto al hacer clic en el elemento de menú File > Load file… y muestra un cuadro de diálogo de información con la cantidad total de páginas web procesadas: 0,5 puntos.
- Iniciar el proceso al hacer clic mediante la opción *Process* > *Start*:
  - Crear una tarea para cada página web que obtenga la lista de enlaces y actualice el número total de enlaces: 2 puntos.
  - Definir un *Executor* y lanzar los hilos: 1 punto.
  - Verificar el *executor* y actualizar la vista con un *ScheduledService* (incluida la obtención de los objetos *WebPage* cuando todo termine): 2,5 puntos.
- Mostrar los enlaces en la lista de la derecha cuando se hace clic en una página web: 0,5 puntos.
- Mostrar mensajes de error siempre que no se pueda hacer algo (por ejemplo, iniciar un proceso sin cargar el archivo previamente): 1 punto.
- Documentación del código (comentarios de *Javadoc* para cada clase y método público o constructor), limpieza y eficiencia: 1 punto.

5.2. Acerca de las mejoras opcionales

Los cambios opcionales propuestos pueden mejorar tu nota hasta 2 extra puntos:

- Si obtienes menos de 9,5 puntos en la parte obligatoria, la nota máxima será de 10.
- Si obtienes al menos 9,5 puntos en la parte obligatoria, la nota máxima será de 11.