UNIVERSITAT DE BARCELONA

Ingeniería Informática

Programación II

Reproductor Multimedia

Práctica 1 - *25/02/18*

Nombre	**NIUB**
Manuel Ernesto Martínez Martín	20081703
Aarón Peruga Ortiga	20026941

Índice

Introducción

El objetivo de la práctica es hacer el modelo (carpeta de ficheros y ficheros) y la Vista por consola de la aplicación que su finalidad es ser un Reproductor Multimedia.

Utilizamos: una librería para ayudarnos en la creación del menú, encapsulamiento, herencia y las librerías io y util (para practicar ArrayList) de java para desarrollar la aplicación.

Análisis

Hemos pensado que contando la clase Menu de la librería **UtilsProg2.jar**, tendremos 6 clases en el ejercicio.

Tendríamos:

1. La clase compilada Menu.

2. Una clase para el main de la aplicación, que será el centro inicial de ésta.

3. Una clase para definir un fichero multimedia, atributos, getters, setters y toString.

4. Una clase contenedora de ficheros multimedia, para poder gestionar el catálogo, con métodos para operaciones CRUD, comprobaciones y un toString.

5. Una clase excepción para cuando la carpeta de ficheros está llena o el nombre del fichero no es válido (nos parece buena idea para practicar las excepciones).

6. Una clase para añadir el menú (no usar en el main para no sobrecargar y no incumplir tanto los principios SOLID, aunque incumpliremos unos pocos hasta que la aplicación esté totalmente terminada), como dice en el enunciado quedará instanciado un objeto de dicha clase en el main.

Desarrollo

Siguiendo el enunciado completo del pdf hemos llegado a que:

1. En el enunciado decía que el tamaño máximo del ArrayList de la carpeta de ficheros tenía que ser 100 y que se lo escribiéramos como parámetro al crear el ArrayList → **ArrayList<FitxerMultimedia> = new FitxerMultimedia(100)**. Pero ese parámetro no es el tamaño máximo permitido, es el tamaño inicial, al final el tamaño máximo lo hicimos con una comprobación a la hora de añadir.

2. Respecto a algunos Setters hemos pensado que no deberían estar (nombre del fichero, extensión, ruta, fecha de última modificación) puesto que dependen de la ruta de la clase madre File y ésta es inmutable, si dejamos settear cualquier cosa podría no cuadrar con el lugar y el nombre del fichero, así mismo también pensamos que no deberían ser atributos y directamente ser Getters.

3. Hemos añadidos a fin de facilitar la clase **FitxerMultimedia** una serie de métodos que hemos usado en conjunto con los getters:

    a. **findName()**: para recoger el nombre sin extensión del fichero.
    b. **findPath()**: para recoger la ruta absoluta sin el fichero.
    c. **findExtension()**: para recoger la extensión del fichero.

4. Hemos pensado que a la hora del usuario elegir entre 0 y size()-1 para borrar un fichero, sea entre 1 y size() (restando 1 internamente).

5. En el apartado de “camino completo” del toString de FitxerMultimedia, hemos usado la ruta absoluta completa tal y como está en el ejemplo del pdf.

6. Hemos usado StringBuilder para hacer más óptimos los Strings que hay que concatenarlos mucho.

El menú en la opción de añadir un fichero no lo hemos hecho como en el pdf de ayuda, puesto que como usamos el constructor heredado de File con el parámetro ruta, debemos crear el fichero antes de pedir los datos, entonces los hemos puesto dentro del case del menú.

Cuestiones Planteadas

Hemos seguido todos los puntos que indica el enunciado para escribir el programa usando las mismas librerías y métodos con algún añadido extra ya comentado como: Excepciones, StringBuilder.

También decir que hemos interpretado que aunque estén en diferentes carpetas los ficheros, corresponden a la misma carpeta ficticia de ficheros, esto viendo los ejemplos del pdf y que CarpetaFitxers no tiene un atributo por así decirlo que le identifique como una carpeta en concreto.

Clases Implementadas

1. **IniciadorAplicacioUB**: Es la clase principal del programa y la que contiene el main.

2. **AplicacioUB1**: Es la clase que contiene el menú y la carpeta de ficheros, se encarga de añadir ficheros a la carpeta, mostrarlos, borrarlos y salir de la aplicación.

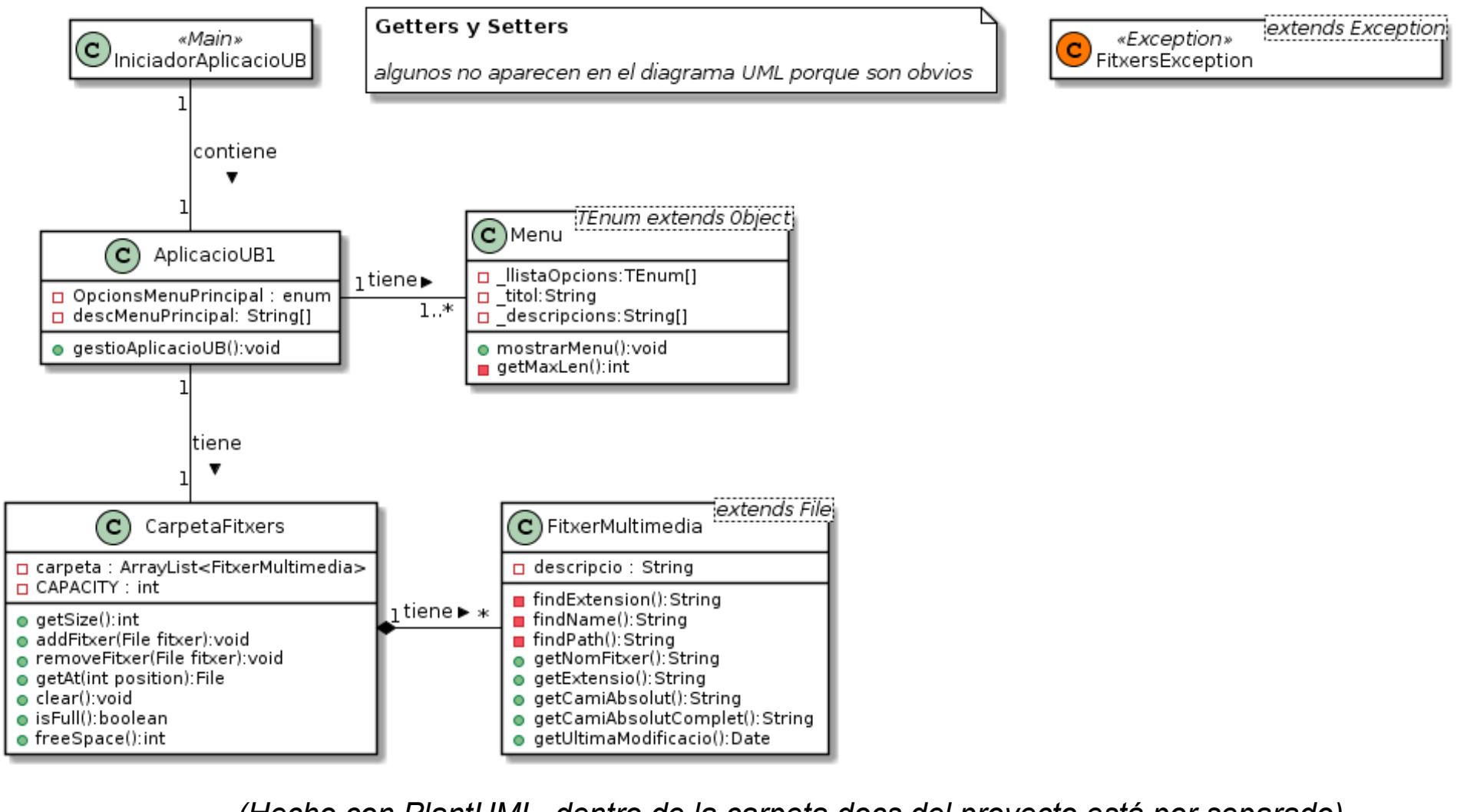
3. **Menu**: Es la clase proveniente de la librería **UtilsProg2.jar**, tiene los atributos y los métodos para usar un menú rápidamente.

4. **CarpetaFitxers**: Es la clase que se encargará de almacenar y destruir los ficheros multimedia en su lista.

5. **FitxerMultimedia**: Extiende de File, Es la clase que usaremos para asignar un fichero a ésta.

6. **FitxersException**: Excepción usada cuando se intenta añadir un fichero y la carpeta de ficheros está llena, cuando un nombre no es válido, etc.

Diagrama de Clases



(Hecho con PlantUML, dentro de la carpeta docs del proyecto está por separado)

Implementación de isFull

Hemos creado una constante CAPACITY de valor 100 y un método isFull() que retorna un boolean dependiendo si el size() de la carpeta es igual a CAPACITY.

Después usamos ese método en un if dentro del método addFitxer(), si es True lanzará una excepción que hemos tratado y mostrará un mensaje de que la carpeta está llena, si es False añadirá el fichero si además cumple la condición de que no está dentro de la carpeta de ficheros.

Implementación de CarpetaFitxers

Respecto a cuando hay dos ficheros iguales y al método removeFitxer(), es imposible que tengamos dos ficheros iguales, puesto que en el pdf de la práctica indicaba que había que hacer una implementación del método equals() dónde se decía que si todos los atributos eran iguales los dos ficheros sería el mismo.

Dicho esto jamás dejará añadir un fichero igual puesto que hay un if antes de añadir que ejecuta el método contains() de ArrayList.

Resultados

Hemos hecho test unitarios con JUnit en la clase CarpetaFitxers (CarpetaFitxersTest), ya que era la única clase que podía petar debido al control de ficheros, los test los hemos pasado satisfactoriamente:

1. Se añaden bien los ficheros.
2. Comprobando que evidentemente si hay un fichero igual a otro no lo añade.
3. No añade más archivos si hay 100.
4. Se eliminan los archivos bien con los índices empezando desde 1.