REPRODUCTOR MULTIMÈDIA

UNIVERSITAT DE BARCELONA

Blai Ras Jimenez

Programació II

Lliurament 1

20/03/2016

Índex

1. Introducció

El projecte de programació II d'aquest any és realitzar un reproductor multimèdia, és a dir, un software que permeti guardar, gestionar i reproduir diferents arxius multimèdia com fotografies, vídeos, cançons o textos.

En la primera entrega, comencem a dissenyar com funcionarà aquest reproductor. Es tractarà doncs de definir un menú amb les opcions que permetrà el nostre reproductor, i implementar aquelles de més senzilles.

Així doncs, en la primera part del projecte es planteja realitzar les 3 primeres opcions del que seria el nostre primer menú, molt primitiu encara. Aquestes 3 opcions són les bàsiques en un reproductor: afegir un arxiu, eliminar algun arxiu i mostrar tots els arxius.

Per tant, en la primera entrega no reproduirem cap tipus d'arxiu, sinó que programarem una primera gestió de com administrar els diferents arxius que ens puguin afegir. Per fer-ho, utilitzarem alguns conceptes que ja em vist en Programació I, com son les taules o més en general la Programació en objectes.

Un fitxer doncs quedarà representat per uns certs atributs fixes: nom del fitxer, extensió, ruta d'on es troba, data d'última modificació i nom desitjat pel fitxer. Els atributs de nom, ruta i data estaran dins la superclasse File, de la qual per tant heretarem més d'un mètode.

Es planteja que el conjunt de fitxers s'implementi de dos maneres diferents; amb una taula de fitxers normal de tipus File o amb un ArrayList de FitxersMultimedia. El primer mètode ja el coneixem de Programació 1, però els ArrayList són nous.

Per l'altra banda, tenim l'opció d'eliminar un fitxer, que té com a objectiu que l'usuari seleccioni un fitxer a eliminar mitjançant el seu índex, de manera que tant la taula com l'Array haurien de quedar sense espais en blanc una vegada eliminats.

Per últim, per mostrar tots els arxius afegits es demanen els mètodes toString() ja vistos a Programació I, i tenen com objectiu imprimir per pantalla totes les propietats ja citades de cada fitxer afegit.

En la segona part de la primera entrega, es té com objectiu implementar les opcions 4 i 5 del menú: guardar una carpeta o recuperar una carpeta. Guardar carpeta ens crearà un arxiu en el nostre disc amb la informació dels arxius afegits, mentre que

recuperar carpeta agafa aquest arxiu amb la informació dels fitxers i ens la carregarà al nostre reproductor.

Per fer-ho, s'utilitzaran els Streams de Java, una espècie de canals per on podem obtenir o enviar informació d'objectes. Aquests, ens permetran escriure en un fitxer o llegir informació d'un, i d'aquesta manera guardar/recuperar carpetes.

2. Anàlisi

El reproductor multimèdia es divideix en tres mòduls principals: Vista, Model i Controlador. En la primera entrega però, només s'implementarà Vista i Model.

Vista es el mòdul que mostra les dades i funcionalitats del reproductor, conté les la classe "Main" i fa possible la interacció amb l'usuari, en aquest cas, amb el menú del reproductor.

Model en canvi es el mòdul que guarda les dades que es mostren en Vista. Es per tant on es troben les classes que guarden, recuperen o eliminen arxius i que contenen tots els mètodes necessaris per dur a terme les instruccions que s'envien des de Vista.

Controlador no s'implementa en la entrega 1, però controla l'execució del programa i gestiona la informació privada que s'envia entre Vista i Model.

3. Explicació de les classes implementades

- Primera part:
    - ***AplicacioUB1:*** és la classe que integra el mètode gestió() que conté el menú amb les diferents opcions del reproductor. Consta de 6 opcions, en la que cada una es un case. En cada cas no s'implementa el codi que es selecciona l'opció, sinó que es criden els mètodes implementats amb els paràmetres necessaris. Durant la primera part, només té un atribut taula de tipus TaulaFitxers.
    - ***IniciadoraplicacioUB:*** classe que conté el mètode Main del nostre projecte. Només crea una nova AplicacioUB1 per poder cridar el mètode gestió() amb el nostre menú.
    - ***FitxerMultimedia:*** és la classe que conté tota la informació sobre un fitxer, per això hereta de la superclasse File. Només conté un atribut, i és el nom que jo li desitjo posar a un arxiu. Conté també tots els getters per obtenir la informació dels atributs de tot arxiu. També té el mètode toString() que permet imprimir la informació d'un arxiu.
    - ***TaulaFitxers:*** la classe que crea i gestiona la taula de fitxers del nostre reproductor. Té per tant els mètodes que permeten afegir o eliminar

un fitxer, i d'altres que ajuden en la seva implementació, com saber si la taula està plena o buida o obtenir un fitxer per la seva posició.

- Segona Part:
    - AplicacioUB1: ara aplicació UB1 té un constructor que rep per paràmetre un booleà, el qual decidirà si s'implementa el reproductor mitjançant TaulaFitxers o mitjançant l'ArrayList de CarpetaFitxers. Ara l'únic atribut que té es de tipus InFileFolder. Les altres diferències són únicament la implementació dels mètodes guardarCarpeta i recuperarCarpeta, que corresponen a les opcions 4 i 5 del menú.
    - IniciadorAplicacioUB: ara el Main el primer que fa es preguntar-te per pantalla si desitges utilitzar CarpetaFitxers o TaulaFitxers, per després cridar a AplicacioUB1 segons la decisió presa.
    - CarpetaFitxers: és la classe que conté l'ArrayList de fitxers. És per tant semblant a TaulaFitxers, però aquest cop fem Override de Serializable i InFileFolder, d'on utilitzem els mètodes ja fet de addFitxer i removeFitxer. Conté també altres mètodes d'ajuda com isFull() o getAt(), i un toString per imprimir per pantalla la informació d'un arxiu.

Per tant, quan he creat la classe CarpetaFitxers he hagut de tenir en compte la classe TaulaFitxers, ja que fan el mateix. En conseqüència, he aprofitat mètodes que ja tenia fets, com isFull() o isEmpty(). Altres mètodes han patit petites modificacions, com getSize(), que en comptes de retornar un int numFitxers, m'he limitat ha cridar el mètode .size() de la superclasse InFileFolder. El mateix passa amb getAt(), ja que crido el mètode .get() en comptes d'utilitzar la sintaxis taula[posició].

4. Desenvolupament

- Primera Part:

  En la primera part ens disposem a realitzar les opcions 1, 2 i 3 del menú, però primer de tot haurem de crear aquest menú. Per fer-ho, disposem d'un menú de mostra en el campus, de manera que seguirem el seu esquema. El menú ha d'estar implementat en la classe AplicacioUB1, ja que es la encarregada de "posar en marxa" el reproductor. Per fer-ho, disposem d'un mètode gestioAplicacioUB(), on declararé aquest menú.

  Començo declarant els atributs del menú: un String amb el títol de les opcions i el nom dels diferents case. Un cop inicialitzat, creem el switch d'opcions, en la que la variable opció serà un Scanner que es crida cada vegada a no ser que es desitgi sortir del menú.

  El primer case doncs afegirà fitxers al reproductor. Per fer-ho, simplement creo un nou arxiu de tipus FitxerMultimedia. Per crear-lo, necessito saber la ruta d'aquest, de manera que la demano per teclat. També demano per pantalla el nom personal que desitgí posar-li al fitxer, amb l'ajuda del mètode demanaDadesTeclat() que veurem després. Un cop creat l'arxiu, faig un try { d'afegir el fitxer a la taula, ja que si alguna cosa surt malament, disposo d'un catch amb l'excepció corresponent:

```
case MENU_PRINCIPAL_AFEGIRF: //Opció d'afegir un fitxer
    System.out.println("Has triat Afegir Fitxer");
    //Demano el cami on es troba el fitxer a afegir
    System.out.println("On està el fitxer que vols afegir?");
    String cami;
    cami = sc.nextLine();

    arxiu = new FitxerMultimedia(cami); //Creo l'arxiu amb el cami demanat
    //Actualizto els arxius amb tota la informacio que em demana l'enunciat

    arxiu.demanaDadesTeclat(); //Demano per teclat el nom que ell desitja

    try { //Intenta, si no sorgira cap error...:
        taula.addFitxer(arxiu); //Afegeixlo
        System.out.println("Fitxer afegit correctament!");
    } catch (AplicacioException error){
        //Llença l'error que hagi pogut ocorrer
        System.out.println("Fitxer NO afegit correctament, " +error.getMessage());
    }
    break;
```

El segon case és el corresponent a eliminar un fitxer. Per fer-ho, demano l'índex del fitxer a eliminar, que pot saber mostrant la carpeta en l'opció de menú 3. Així doncs, l'usuari introdueix el número del fitxer de manera que puc cridar el mètode removeFitxer() amb el paràmetre taula.getAt(index-1). El mètode getAt() el veurem després però obté un fitxer mitjançant el seu índex. Li resto 1 al índex ja que els Arrays en Java comencen per la posició zero, mentre que jo començo a contar arxius a partir del número 1.

Abans d'eliminar un fitxer, però, comprovo si la taula està buida.
El tercer case correspon a mostrar tota la informació del arxius continguda en la taula. En conseqüència, hauré de recórrer tota la taula amb un for i per cada arxiu cridar el seu mètode toString(), del qual parlarem més tard. Abans però, comprovo si la taula conté fitxers! (En la segona part, es treu el "for" per un simple System.out.println(taula))

El menú però és inútil si no tenim les classes implementades. La primera classe que faig és FitxerMultimedia, ja que és la base de qualsevol fitxer a afegir. FitxerMultimedia és una classe que "extends" File, és a dir, utilitza i modifica mètodes de la superclasse File generada per Java. Ens demanen els següents mètodes:

- getNomFitxer(): getter que retorna el nom del fitxer que l'usuari assigna.
- setNomFitxer(String nom): setter que defineix el nom d'un arxiu que l'usuari desitja.
- getNom: getter que retorna el nom original d'un fitxer.
- getUltimaModificacio(): getter que retorna la Data de l'última modificació de l'arxiu afegit.
- getCamiAbsolut(): getter que retorna el camí on es troba el fitxer.
- getExtensio(): getter que retorna l'extensió del fitxer. Per fer-ho, es fa servir el mètode lastIndexOf(), que retorna l'índex del caràcter passat per paràmetre del String (en aquest cas, l'string es el camí, i el caràcter el punt).
- Boolean demanaDadesTeclat(): mètode que demana per pantalla el nom desitjat per un arxiu, i després de comprovar si es correcte, l'assigna al arxiu que es vol afegir.

- Boolean equals(Object fitxerMultimedia): booleà que comprova si dos arxius són el mateix.
- toString(): mètode que retorna un String que conté tota la informació d'un arxiu: el nom, la data, el nom del fitxer assignat, la extensió i el camí del arxiu.

Per tant, FitxerMultimedia només té un atribut: nomQueliposo, és a dir, l'únic atribut que assigna l'usuari. La resta, farem servir la sintaxis super.(mètode) de la classe File per obtenir la informació desitjada. És el cas de getNom(); getCamiAbsolut() i equals(). Ho podem veure per exemple amb getNom:

```
public String getNom() { //Metode que troba el nom original del fitxer dins la clase File
    return super.getName();
}
```

Un cop definit el que serà un arxiu, definirem on estaran guardats: es tracta doncs de la classe TaulaFitxers, que no és més que una taula de tipus File. Com que guarda fitxers, la classe TaulaFitxers "implementa" InFileFolder, una superclasse generada per Java que conté mètodes que ens interessa modificar o utilitzar.

Ens demanen els següents mètodes:

- getSize(): retorna el número de fitxer afegits, que en aquest cas està representat per un int anomenat numFitxers que s'incrementa en afegir un arxiu.
- addFitxer(File fitxer): aquest mètode afegeix un arxiu a la taula. Retorna una excepció de tipus AplicacioException, si la taula està plena. Per afegir un fitxer doncs, comprova si la taula està plena, i si hi ha espai, afegeix el fitxer passat per paràmetre en la següent posició de la taula disponible. Després, incrementa el int numFitxers.
- removeFitxer(): aquest mètode elimina un fitxer. Per fer-ho, comprova fitxer per fitxer de la taula amb el fitxer passat per paràmetre, amb l'ajuda del mètode equals() que veurem a continuació. Si es dona la coincidència, assigna "null" a l'espai de la taula on es troba el fitxer, és

a dir, l'elimina. En el cas que es doni la eliminació d'un arxiu, amb un for elimino aquests espais en blanc "null" que s'han pogut donar a la taula, i ho faig mitjançant els índex d'ella: cada posició i-1 serà i.

- Clear(): aquest mètode elimina tots els fitxers de la taula. Per fer-ho, es recorre tota ella i per cada posició se li assigna un "null".
- getAt(): mètode important! Retorna un fitxer de la taula corresponent al índex passat per paràmetre.
- isFull(): booleà que comprova si la taula està plena. Ho faig mitjançant l'enter que em compta arxius: return (numFitxers == 100).
- isEmpty(): semblant a isFull(), però retorna cert si la taula està buida.
- String toString(): mètode encarregat de recorre la taula i per cada arxiu imprimir la seva informació per pantalla, mitjançant el toString() de TaulaFitxers. El for per tant segueix l'esquema "for (int i = 0; i < numFitxers; i++)" i dins l'String en qüestió més taulaFitxers[i].toString().

Ara només queda la Classe més important de totes, la que decidirà construir el reproductor: IniciadorAplicacioUB. És aquella classe que conté el mètode main. Aquest mètode simplement accionarà a AplicacioUB1, i ho farà simplement creant una nova aplicació: AplicacioUB1 aplicacio = new AplicacioUB1(); i seguidament cridarà el nostre menú per poder utilitzar el nostre reproductor: aplicació.gestioAplicacioUB().

- Segona Part:
  En la segona part de la primera entrega es planteja realitzar les opcions 4 i 5 del menú i ha agrupar els arxius en un ArrayList, i no en una senzilla taula. Les opcions 4 i 5 corresponen a guardar i recuperar una carpeta, però primer començarem explicant la nova classe que tenim a Model: CarpetaFitxers.

  Com el seu nom indica, CarpetaFitxers es la classe que implementa una carpeta on es guardaran els fitxers afegits en el nostre reproductor. Per tant, la seva funcionalitat i el seu objectiu és exactament el mateix que TaulaFitxers, però en comptes d'utilitzar una taula, utilitzarem un nou concepte de Java: els ArrayList.

  CarpetaFitxers per tant modifica mètodes que es troben a InFileFolder, al igual que TaulaFitxers. El primer que contindrà aquesta classe serà l'ArrayList ja mencionat. Per declarar un ArrayList, fem servir la sintaxis ArrayList<tipus>nom, és a dir, amb "ArrayList<FitxerMultimedia>taulaFitxers estic declarant un ArrayList de objectes tipus FitxerMultimedia anomenat taulaFitxers. El màxim de fitxers en aquest cas continua sent 100, i es declara com a atribut de classe i s'inicialitza en el seu constructor:

```
public class CarpetaFitxers implements InFileFolder { //Carpeta Fitxers modifica de la classe InFileFolder

    protected ArrayList<FitxerMultimedia>taulaFitxers; //Declaro l'ArrayList

    public CarpetaFitxers() {
        taulaFitxers = new ArrayList<FitxerMultimedia>(100); //Defino l'arrayList

    }
```

Els mètodes que conté aquesta classe són molt semblants als de TaulaFitxers:

- getSize(): getter que em retorna la mida del Array. En comptes de fer us d'una variable que s'incrementa en afegir un fitxer, ara podem utilitzar el mètode de InFileFolder anomenat .size() que retorna aquest int corresponent a la mida.
- addFitxer(): mètode que s'encarrega d'afegir un fitxer a l'ArrayList passat per paràmetre. Per fer-ho, fem us d'un altre mètode que trobem a InFileFolder: .add(). Primer de tot però, comprovem si la taula està plena, i per tant implemento la excepció corresponent, és a dir, un missatge d'error en el cas d'intentar afegir fitxers amb la taula plena.
- removeFitxer(): mètode que s'encarrega d'eliminar un fitxer passat per paràmetre. Per eliminar-lo, faig us del mètode de InFileFolder .remove(), de manera que no haig d'esborrar un fitxer amb "null". Per indicar quin fitxer estic eliminant, en creo un de nou passant-li el camí com a paràmetre, així m'estalvio un cast de FitxerMultimedia:
- getAt(): getter que em retorna el fitxer corresponent al índex passat per paràmetre. Per fer-ho, utilitzo un altre cop un mètode de la classe InFileFolder: .get().
- Clear(): mètode que elimina tots els arxius de l'ArrayList. Per fer-ho, utilitzo el mètode de la classe InFileFolder .clear().
- isFull(): booleà que retorna "cert" si la taula està plena, i "fals" si no ho està. Utilitzo el mètode getSize() igualat a 100 com a booleà, és a dir, si el número retornat de getSize() és 100, retorna true, és a dir, la taula està plena. Si no, retorna false.

- String toString(): mètode que s'encarrega de mostrar la informació d'un fitxer per pantalla. Per recorre la taula, utilitzo els Iteradors de Java, de manera que no em fan falta el "for" de TaulaFitxers. Per iterar la meva taula faré us doncs del mètode Iterator, que declaro amb la sentencia Iterator itrFitxers = taulaFitxers.iterator(). D'aquesta manera, podré fer un while de "mentre la taula tingui més arxius", és a dir, while (itrFitxer.hasNext()), retorna un String del fitxer en qüestió, que aconseguiré gràcies al altre mètode toString() de FitxerMultimedia juntament amb el seu índex, que no es més que un enter que es va incrementant:

```
@Override
public String toString() { //Mètode que m'imprimeix per pantalla tots els fitxers de la taula
    Iterator itrFitxer = taulaFitxers.iterator();
    String s = "\n------------------TAULA FITXERS------------------\n";
    FitxerMultimedia file;
    int index = 0;
    while (itrFitxer.hasNext()) {
        file = (FitxerMultimedia) itrFitxer.next();
        index++;
        s = s+ "["+index+"]"+taulaFitxers.toString();
    }
    return s;
}
```

Un cop tenim CarpetaFitxers realitzat, haurem de fer alguna cosa per poder triar entre un reproductor multimèdia implementat amb una Taula de Fitxers o amb una Carpeta de Fitxers. La solució a aquest problema es planteja modificant el constructor de AplicacioUB1, el qual rebrà per paràmetre un booleà amb la informació necessària per crear una taula de fitxers de tipus CarpetaFitxers o de tipus TaulaFitxers. Per tant, l'atribut "taula" haurà de ser de tipus InFileFolder, ja que suporta els dos tipus d'implementació. Quedaria més o menys així:

A part d'això, AplicacioUB1 es modifica també en la segona entrega fent les opcions 4 i 5 del menú. La primera opció, la 4, és l'encarrega de guardar el conjunt de fitxers en un arxiu "físic" en el nostre disc dur. És a dir, al seleccionar la opció 4, es crearà un arxiu en una ruta definida que contindrà la informació dels arxius afegits.

L'encarregat de fer-ho serà el mètode guardarCarpeta(File fitxer), implementat en la classe AplicacioUB1 mateix. Aquest mètode rep per paràmetre un fitxer el qual s'encarregarà de guardar. Per poder establir aquesta "relació" entre el nostre reproductor i el nostre ordinador en sí farem us dels Streams de Java, una eina que permet crear "canals" d'entrada i sortida per on es transmetran aquests arxius a guardar. En conseqüència, fem l'import de la superclasse Serializable que conté els mètodes FileInputStream, ObjectInputStream i ObjectOutputStream.

Primer de tot, declarem un canal de sortida "fout" amb el paràmetre "fitxer" que passem precisament per paràmetre del mètode: "FileOutputStream fout = new FileOutputStream(fitxer)". Seguidament, creem el canal de sortida dels fitxers, anomenat "oos", que té com a paràmetre precisament aquest canal de sortida d'informació "fout". Un cop tenim les vies necessàries per enviar la informació, fem us del mètode writeObject() que conten els

```
public AplicacioUB1(boolean impliTaula) {
    if (impliTaula) { //Si es cert que vull una implementacio amb una taula...:
        taula = new TaulaFitxers();
    } else {
        taula = new CarpetaFitxers();
    }
}
```

ObjectOutputStream que és l'encarregat "d'escriure" aquesta informació d'arxius a guardar dins d'un nou fitxer, que acostuma a tenir l'extensió .dat tot i que també pot ser un .txt. Ens en recordem

també sempre de tancar aquest canals, pel bon funcionament del programa, amb el mètode .close().

```
public void guardarCarpeta(File fitxer) {
    try(FileOutputStream fout = new FileOutputStream(fitxer)) {
        ObjectOutputStream oos = new ObjectOutputStream(fout);
        oos.writeObject(taula);
        oos.close();
        fout.close();
    } catch (FileNotFoundException ex) {
        System.out.println("Error, el fitxer no existeix!");
    } catch (IOException ex) {
        Logger.getLogger(AplicacioUB1.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Tot aquesta implementació, però, està continguda en un format “try-catch”, per poder gestionar les possibles excepcions que poden sortir. En aquest cas, controlo dos excepcions: FileNotFoundException, que avisa a l’usuari de la inexistència del fitxer a guardar, i IOException, que avisa d’un error d’interrupció d’operacions I/O:

La cinquena i última opció, recuperar carpeta, en canvi, el que fa és recuperar arxius afegits al reproductor anteriorment llegint la informació d’un fitxer prèviament guardat gràcies a la opció quarta del menú, guardar carpeta.

Per tant, el mètode recuperarCarpeta(File fitxer) també crearà Streams, però seran de tipus “entrada”. Rep com a paràmetre un fitxer, que en aquest cas és el que conté la informació a recuperar.

El primer que farem doncs és preguntar per pantalla el camí on es troba aquest fitxer, de manera que podem crear un nou FileInputStream amb el paràmetre “arxiu a recuperar” que hem pogut crear gràcies al camí entrat.

Un cop tenim la via per recuperar informació, creem el Stream per on entraran els fitxers que s’afegiran al reproductor. Ho fem de manera semblant a guardar Carpeta, però aquest cas serà de tipus Input:

ObjectInputStream ois = new ObjectInputStream(fin);

On “fin” és el Stream de fitxers creat anteriorment. Per llegir la informació, fem us del mètode ois.readObject(). Tot i això, necessitem un cast de InFileFolder ja que estic llegint informació de tipus superclasse Object, i jo necessito que aquest Object sigui de tipus InFileFolder per poder-la afegir.

Aquest mètode també necessita estructura “try-catch”, per poder controlar les excepcions de ClassNotFoundException i l’anterior IOException. Ens en recordem també de tancar els canals d’entrada:

```
public void recuperarCarpeta(File fitxer) {
    Scanner sc;
    sc = new Scanner(System.in);
    String camiRecuperar;
    System.out.println("A on has guardat la carpeta?");

    camiRecuperar = sc.nextLine();
    fitxer = new File(camiRecuperar);
    try {
        FileInputStream fin = new FileInputStream(fitxer);
        ObjectInputStream ois = new ObjectInputStream(fin);
        taula = (InFileFolder)ois.readObject();
        ois.close();
        fin.close();
    } catch (IOException ex) {
        Logger.getLogger(AplicacioUB1.class.getName()).log(Level.SEVERE, null, ex);
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(AplicacioUB1.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

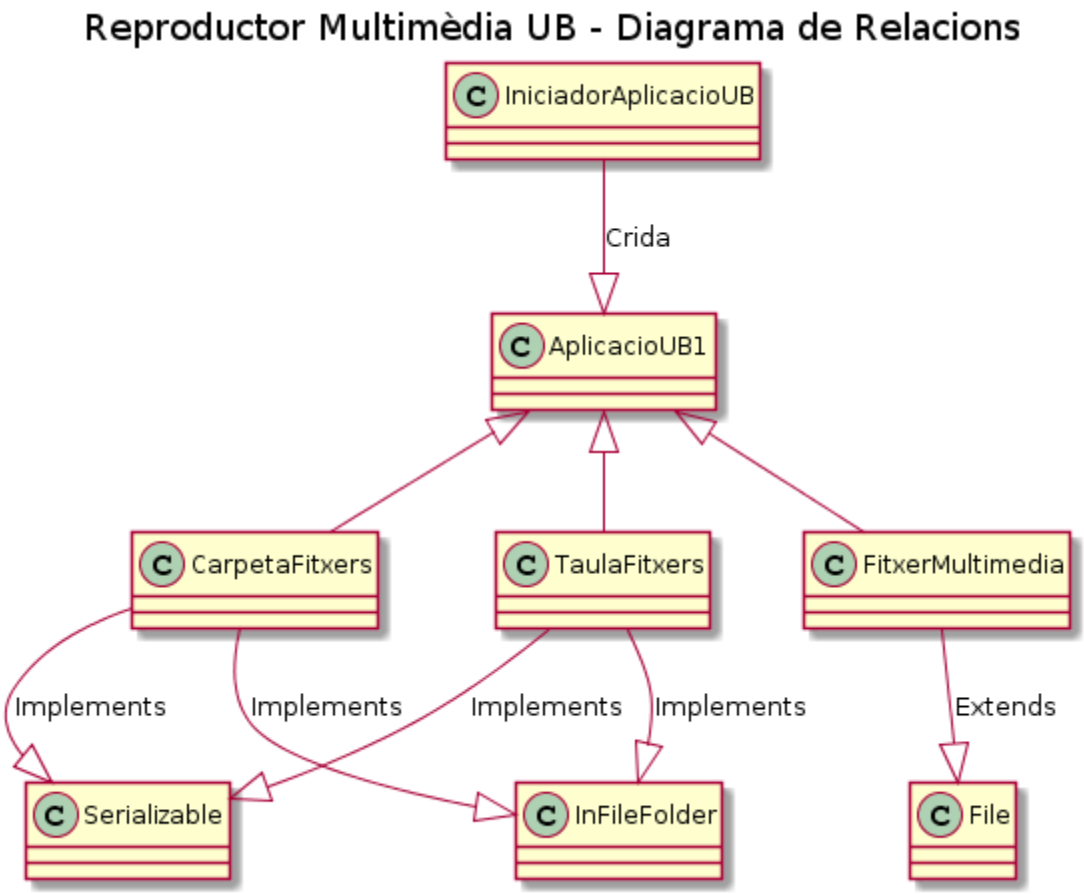
Per acabar, recordem que he modificat el constructor de AplicacioUB1 de manera que rebi un booleà, per tant modificaré també en aquesta segona part la classe IniciadorAplicacióUB. Concretament, el mètode main en aquest cas és l'encarregat de preguntar al usuari per pantalla si vol una implementació de tipus CarpetaFitxers o de tipus TaulaFitxers.

Per fer-ho, simplement gestiono la resposta amb un if, que modificarà el booleà a "transmetre", i després cridarà a AplicacioUB1 per "crear" el reproductor:

```
public static void main(String[] args) {
    Scanner sc;
    sc = new Scanner(System.in);
    boolean impliTaula;
    String resposta;
    System.out.println("Vols utilitzar TaulaFitxers o CarpetaFitxers? (T/C)"); //Pregunto quina implementacio vol
    resposta = sc.nextLine(); //La proceso
    if ("T".equals(resposta)) { //Si la resposta es "T"...:
        impliTaula = true;
    } else {
        impliTaula = false;
    }

    AplicacioUB1 aplicacio = new AplicacioUB1(impliTaula); //Crea el reproductor

    aplicacio.gestioAplicacioUB(); //Administra el reproductor

}
```

La classe TaulaFitxers i FitxerMultimedia no es modifiquen en aquesta segona part.

En definitiva, el nostre reproductor en Java basat en la POO tindria la següent relació de classes en la primera entrega:



5. Respostes

En aquest apart responc les preguntes proposades en l'entrega 1 que no he respost durant el desenvolupament. Són les següents:

- *Segons la implementació de la classe CarpetaFitxers, si tenim dos fitxers multimèdia corresponents al mateix fitxer, quan cridem al mètode per eliminar un d'aquests fitxers eliminarà l'altre també o no?*
    - No, ja que jo demano un índex, el qual faig servir per poder passar per paràmetre el fitxer en qüestió amb el mètode getAt(index). D'aquesta manera, passo un fitxer en concret, i per molt que hi hagi dos d'iguals, aquests dos mai tindran el mateix índex, ja que estan en posicions de la taula diferents.
- *Observacions generals.*
    - Com a observació, voldria comentar la dificultat de gestionar els toStrings() per poder fer un simple System.out.println(taula) en el case, ja que també podríem fer només un únic toString() a FitxerMultimedia i fer un "for" en el case. He implementat el toString() de CarpetaFitxers amb iteradors i sense, i com que m'agrada més el format de sense iteradors, l'he deixat comentat en el codi.
    - També comentar que m'ha agradat veure l'herència i els "extends" de les diferents classes ja que així és com et fas una idea de com es Java i la POO, cosa que no havia quedat clara a Programació 1, almenys per mi.
    - Fent la memòria, m'he adonat de que tenia declarat que CarpetaFitxers implementaba Serializable, i dic, ui que raro, si ha CarpetaFitxers no faig servir mètodes de recuperar ni guardar Carpeta, deu estar equivocat... I al final, al comprovar que tot anava bé, no podia ni guardar ni recuperar, i fins que no vaig pensar en que havia tret el "implements", no sabia per què no funcionava, si abans si que ho feia i no havia canviat "res".

6. Resultats

He fet comprovacions que abarquen totes les combinacions possibles, i fins on he pogut testejar, cap error en l'entrega.

- He afegit 3 fitxers usant la implementació TaulaFitxers. He comprovat que s'imprimeixen per pantalla. He provat d'eliminar el del mig, de tornar-lo a afegir, eliminar el primer, tornar-lo a afegir, eliminar el tercer, i fins aquí cap problema.
- Provo ara de guardar la carpeta. Error! Em falta l'implements de Serializable en la declaració de la classe. Solucionat. Guardo la carpeta, surto, recupero

la carpeta, imprimeixo els fitxers i comprovo que s'han recuperat perfectament.

- Comprovo d'afegir dos fitxers iguals i eliminar un d'ells, per contestar a la pregunta proposada. Confirmo la meva teoria.
- Faig el mateix però usant implementació d'ArrayList, mateix tipus de probes. Cap error.