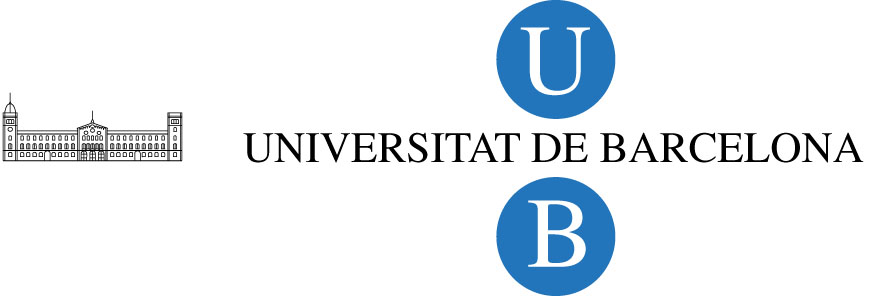
REPRODUCTOR MULTIMÈDIA



Blai Ras Jimenez

Programació II

Lliurament 2

10/04/2016

Índex

1. Introducció
2. Explicació de les classes implementades
3. Desenvolupament
4. Preguntes
5. Introducció

L’objectiu de la pràctica és organitzar fitxers dins una biblioteca que ja prenen el format

L’objectiu de la pràctica és organitzar d’una manera més extensa els nostres fitxers i començar a “reproduir-los” per primera vegada.

Realitzarem dos noves estructures de tipus CarpetaFitxers; Biblioteca i Albums. Una biblioteca es un Array que conté FitxersMultimedia els quals comprova la seva existència en el disc i que no puguin estar repetits. En canvi, un àlbum és una CarpetaFitxers de que sí permet fitxers multimèdia repetits i a part conté l’atribut títol.

Per poder usar el Reproductor, s’haurà de fer un menú més extens, que es divideix en gestionar àlbums i gestionar biblioteca, amb les seves opcions pertinents, a part de les opcions de la primera entrega de Guardar i Recuperar Dades.

Per tant, el Reproductor es dividirà en tres mòduls clars: Vista, on es veuran les dades i funcionalitats del nostre software, Model, que serà on es guardaran les dades, i Controlador, el qual controla l’execució del software.

1. Explicació de les classes implementades

En aquesta entrega es creen i desenvolupen les següents classes noves:

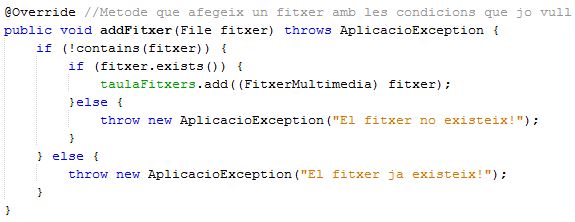
* BibliotecaFitxersMultimedia: contè el conjunt total de fitxers multimèdia sempre i quan compleixin que:
  + No permetrà que hi hagin dos fitxers multimèdia iguals.
  + Quan s’afegeix un fitxer multimèdia a la biblioteca, es verificarà que el fitxer associat existeix a disc.
  + No hi haurà cap límit en el nombre de fitxers multimèdia.
* AlbumFitxersMultimedia: conjunt de fitxers multimèdia, semblant a biblioteca, però amb les condicions següents:
  + Pot tenir fitxers duplicats
  + Hi haurà un màxim de 10 àlbums, però es podrà canviar si l’usuari ho desitja
  + Cada àlbum s’identificarà per un títol
* FitxerReproduible: classe abstracte que gestionarà tots els mètodes relacionats amb reproduir un fitxer Multimedia, juntament amb els atributs d’aquests. Satisfà l’especialització en quan a fitxers reproduïbles (vídeos) i els que no (imatges)
* Vídeo: classe especialitzada de FitxerReproduible. Gestiona la reproducció d’un vídeo, amb els atributs que el defineixen: còdec, alçada, amplada, durada y fps.
* Audio: classe especialitzada de FitxerReproduible. Gestiona la reproducció d’àudios i imatge, amb els atributs que els poden definir, com kilobytes per segon (kpbs) .
* Reproductor: classe encarregada de donar la instrucció de reproduir un fitxer. Com ja hem comentat, aquest pot ser tipus Video o tipus Audio, així que disposa de dos constructors per iniciar la reproducció. Esta continguda dins del paquet Controlador, on també tenim...:
* Controlador: classe encarregada de gestionar l’execució del resta de classes. Tota demanda introduïda per Vista serà transmesa a Controlador qui donarà el vistiplau per accedir a les Dades, les quals es volen amagar de l’usuari.
* CarpetaFitxers: CarpetaFitxers ja està explicada en l’entrega anterior, però la poso aquí per indicar que l’únic canvi que sofreix és en el seu mètode isFull(), que passa a ser sempre fals, ja que no tenim límit ara. A part, implementem el mètode contains() que comprovarà l’existència de fitxers repetits en l’ArrayList
* AplicacioUB2: Aquesta classe és la mateixa que l’anterior, a diferencia, clar, d’un nou menú amb les noves implementacions pertinents, l’eliminació d’un constructor que rep per paràmetre si vull una implementació amb taula o CarpetaFitxers, i un nou mètode imprimir(List<string> llista); el qual s’encarrega de recorre tota la llista que em retornen alguns mètodes d’impressió per pantalla com mostrarBiblioteca(), mostrarLlistatAlbums() o mostrarAlbum();
* FitxerMultimedia: aquesta classe roman exactament igual que l’anterior amb la diferencia d’un constructor amb l’afegit del paràmetre String nom i un this.nom = nom; i per tant l’eliminació del mètode introduirDadesTeclat(); ja que es fa directament al afegir un fitxer i desprès es passa per paràmetre.
* IniciadorAplicacioUB: aquesta classe roman idèntica a la primera entrega, excepte per l’eliminació de la pregunta de si volem una implementació amb carpetaFitxers o taulaFitxers.

1. Desenvolupament

Comencem aquesta segona entrega implementant un nou tipus de carpetaFitxers: BibliotecaFitxersMultimedia. Aquesta classe es per tant un CarpetaFitxers que utilitzarà tots els seus mètodes. Només hi ha una petita diferència: per afegir un fitxer és necessari que aquest existeixi físicament en el disc i que no estigui repetit, és a dir, que no s’hagi afegit anteriorment.

Per fer aquesta comprovació, ens es suficient amb el mètode .exist() que comprova l’existencia del fitxer passat per paràmetre i d’un altre mètode, cotains(), el qual retorna Cert/Fals segons si el fitxer ja existeix a dins seu. Aquest contains() està implementat a la classe CarpetaFitxers(), i fa servir un iterador per recorre tot el seu ArrayList de fitxers afegits els quals compara amb el mètode equals amb el fitxer passat per paràmetre, és a dir, el que es vol afegir.

Per tant, si les dos sentències son certes s’afegirà el fitxer, sinó, es llençarà una nova excepció, la qual pot informar que el fitxer no existeix o que el fitxer està repetit, per això faig els if’s d’aquesta manera:



BibliotecaFitxersMultiemedia per tant s’acaba aquí, ja que totes les seves altres funcions les hereta de CarpetaFitxers. Ara toca implementar el segon tipus de CarpetaFitxers que es demana: AlbumFitxersMultimedia. Aquesta carpeta no serà res més que un àlbum que sí que tenen una mida màxima, 10, i on guardarem fitxers que sí poden estar repetits. Aquesta mida, però, si que es podrà canviar, de fet, es una funció d’aquesta nova entrega.

Cada àlbum estarà identificat per un títol. En conseqüència, implementem els mètodes getTitol() i setTitol(String titol). Com que la mida també es variable, implementem els seus getters i setters: getMida() i setMida(int mida). El seu constructor rebrà aquest títol en qüestió per paràmetre, de manera que jo al crear un nou àlbum estic obligat a posar-li títol i una mida per defecte a 10.

Aquesta mida, es gestionarà amb un mètode isFull(). Aquest isFull() no pot ser el de carpetaFitxers ja que ha estat modificat per retornar sempre fals, ja que BibliotecaFitxersMultimedia no te límit de fitxers. Així doncs, em disposo a fer el mateix mètode però amb @Override, és a dir, modifico el mètode ja existent en carpetaFitxers per retornar cert quan la mida sigui més gran que 10 (o el valor que sigui la mida en aquella ocasió).

Seguidament, realitzo la classe Reproductor, que tot i que no començarem a reproduir de moment, es pot implementar com si ja tinguéssim la resta de classes fetes. Aquesta classe està dins del paquet de controlador, ja que controla l’accés a la reproducció d’un vídeo o audio a part d’accedir al motor de reproducció del VLC Player, el programa que hem escollit per usar el seu motor reproduïble.

Nosaltres no implementarem el codi per reproduir en sí un fitxer, sinó que està implementat en la llibreria que ja vam afegir al nostre projecte UtilsProg2.jar. En ella, trobem la classe abstracte ReproductorBasic, que permet precisament això. En conseqüència, la classe Reproductor “extends” (hereta) de ReproductorBasic.

El seu constructor no es rés més que un accés al camí (path) on es troba instal·lat el VLC Player, el qual passem per paràmetre. Per poder accedir al seu motor, però, nosaltres ja no ho implementem, simplement deixem la feina al mètode públic ReproductorBasic(String vclPatch) fent super(vlcPath).

Els dos mètodes restants es diuen reprodueix, i no són més que l’ordre a reproduir un vídeo. Aquesta ordre la gestionarem amb el mètode .play(), també implementat en ReproductorBasic. Són dos mètodes perquè tal i com explicaré a continuació, no tractem de la mateixa manera un Audio que un Video, ja que un en un Audio també “reprodueixo” una imatge a part clar de la cançó en sí. Així doncs, com que la funció dels dos mètodes és la mateixa, es diuen igual. En el primer, encarregat de reproduir un vídeo (FitxerReproduible) simplement dono delego l’ordre de reproduir el fitxer passat per paràmetre, amb aquest mètode play.

En l’altre mètode, faig el mateix però en aquest cas tinc dos fitxers, el del audio, de tipus Audio, i el de la imatge, de tipus File. Així doncs, delego amb el mètode .play però en aquest cas amb aquests dos paràmetres.

A continuació em plantejo realitzar la classe FitxerReproduible, la qual serà abstracte. És necessari que sigui abstracte ja que deriva d’una especialització dels FitxersMultimedia, és a dir, com que nosaltres afegirem fitxers de tipus Audio i Video, la seva reproducció no serà la mateixa. Per tant, FitxerReproduible “funcionarà” d’una manera o una altre segons si es tracta d’aquests dos tipus d’arxiu, i cridarà al mètode reproduir().

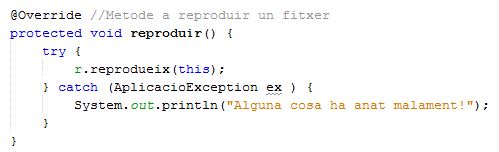
Un cop tinc les eines per reproduir, implemento precisament els dos tipus de fitxerReproduible que podré reproduir: Video i Audio, incloses dins del paquet de Model. Comencem per vídeo!

Vídeo com ja he dit es un tipus de FitxerReproduible, de manera que heretarà els seus mètodes. M’han definit aquests atributs:

* + - * Durada, Amplada i Alçada del Vídeo
      * FPS: frames-per-segon, unitat mínima de temps en que puc reproduir pas a pas el vídeo
      * Nom i camí (Strings)
      * Codec: en quin tipus de còdec està codificat (.MP4, .VLC...)

Així doncs, la seva classe té aquells atributs que són comuns o diferents amb Audio: durada, amplada, alçada, fps i Reproductor r, un atribut de tipus Reproductor necessari. La resta d’atributs, camí, nom, còdec i durada, es delegen a FitxerReproduible, amb el mètode super, clar.

L’altre mètode d’aquesta classe es una modificació del mètode reproduir de FitxerReproduible, que en aquest cas l’adaptarà a les necessitats per reproduir un vídeo. Fem una estructura try-catch per gestionar els possibles errors que puguin succeir:



En Audio, la cosa és molt semblant. El seus atributs de classe són aquells diferents o que no es poden compartir amb Video, en aquest cas, són dos:

* + - * Kpbs: kylobytes per segon, enter que es fa servir sovint per determinar la calitat d’un Audio. Per exemple, 320 kpbs o 256 kpbs en el cas de fitxer d’Audios de iTunes (Apple).
      * File FitxerImatge: File que en aquest cas és una imatge que mostrarem mentre es reprodueix l’Audio en qüestió.

Per tant, en el seu constructor deleguem els atributs de camí, còdec, nom i durada, mentre que assignem (amb this.) la resta d’atributs.

Al igual que Video, també modifiquem a les necessitats d’un Audio el mètode reproduir, el qual també programem amb estructura try-catch. Tot i això, té una única diferència amb Vídeo, i es que cridem al mètode reprodueix amb els paràmetres de this (referintos al Audio) i fitxerImatge per paràmetre, tal i com hem vist en l’explicació de Reproductor.

Ara només hem queden fer dos clases noves: Dades i Controlador. Com que les dos van molt lligades, les explicaré més o menys a la vegada. Controlador és la classe que em garanteix separar les dades i la implementació del usuari, és a dir, una classe que administra l’accés als diferents mètodes i funcions.

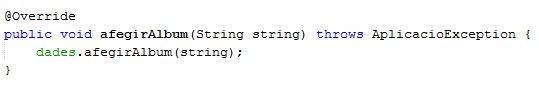
En Controlador, doncs, no hi haurà implementacions de funcions, sinó mètodes que criden a aquestes implementacions perquè es produeixin. És a dir, si jo vull afegir un fitxer, Controlador rebrà l’ordre en qüestió en el mètode en qüestió i cridarà al mètode que afegeix el fitxer.

En conseqüència, té molt mètodes, Controlador:

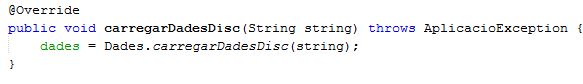
* + - * public void afegirVideo(String path, String nomVideo, String codec, float durada, int alcada, int amplada, float fps);
      * public void afegirAudio(String cami, String camiImatge, String nomAudio, String codec, float durada, int kbps);
      * public List mostrarBiblioteca(): llista dels retorns de toString() dels fitxers
      * public void esborrarFitxer(int id): id és la posició a llista de getBiblioteca()
      * public void reproduirFitxer(int id: id és la posició a llista de getBiblioteca()
      * public void afegirAlbum(String titolAlbum);
      * public List mostrarLlistatAlbums();
      * public void esborrarAlbum(String titolAlbum);
      * public boolean existeixAlbum(String titolAlbum);
      * public void afegirFitxer(String titolAlbum, int id): id és la posició a llista de getBiblioteca() public List mostrarAlbum(String titolAlbum): mostra informació del àlbum
      * public void esborrarFitxer(String titolAlbum, int id): sobrecàrrega del mètode per esborrar un fitxer d’un àlbum. id és la posició a llista de getBiblioteca()
      * public void canviarTitolAlbum(String titolAlbum, String nouTitol);

Aquests mètodes, però, han estat administrats gràcies a l’herència de InControlador, una altre classe abstracte de la llibreria UtilsProg2. La classe té un únic atribut: dades. Aquest, com el seu nom indica, es de tipus Dades, i es que Dades és la classe on està el codi en sí de totes les funcions del nostre reproductor.

Així doncs, per cada mètode de Controlador, simplement cridarem al mètode en qüestió amb aquest atribut. Ho veiem, per exemple, amb el mètode afegirAlbum():

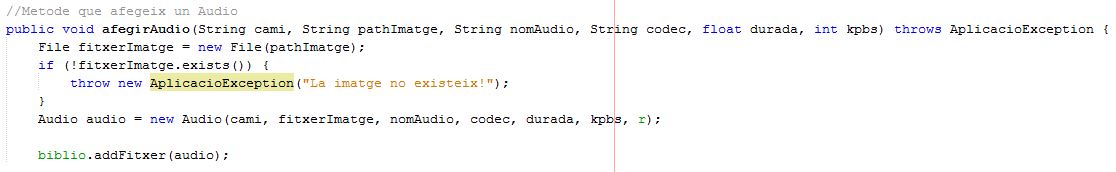


Només hi ha dos mètodes diferents: carregarDadesDisc. Aquest mètode ja vam veure que feia en la primer entrega, però es que ara ja no està implementat en AplicacioUB, sinó que ho està en dades. Com que ara el mètode ois.readObject() ja no pot fer el cast a FitxerMultimedia, faig una implementació (que explicaré més endavant) la qual implica que el mètode sigui static, és a dir, un mètode de classe. Aquest mètode de classe, per tant, implica un altre crida que no pas dades.<mètode>, sinó que faig... :

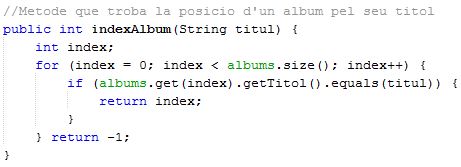


Ara sí, la classe Dades. Dades és la classe que emmagatzema la informació (les dades) de Album i BibliotectaFitxersMultimedia. És a dir, és on estan implementats tots els mètodes que permeten dur a terme totes les funcions que em demanen, com per exemple, canviar la mida d’un àlbum o esborrar un fitxer. Expliquem els seus mètodes:

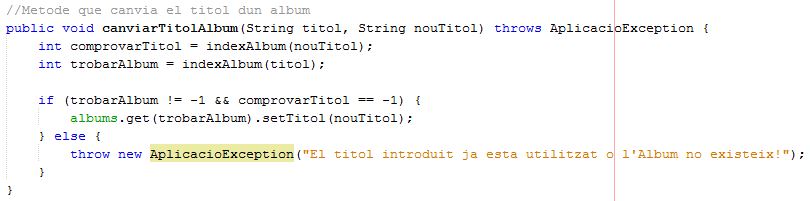
* 1. Dades(): constructor. Inicialitza els atributs album (ArrayList) i bibio (BibliotecaFitxersMultimedia). Tambè inicia l’atribut r de tipus Reproductor.
  2. Void afegirVideo(String camí, String nomVideo, String còdec, float durada, int alcada, int amplada, float fps): mètode encarregat d’afegir un vídeo a la biblioteca. Ho fem primer de tot creant l’objecte vídeo amb els atributs passats, el qual desprès afegirem amb el mètode .addFitxer a la biblioteca, dins d’una estructura try-catch en la que controlem qualsevol excepció que es pugui produir.
  3. Void afegirAudio(String cami, String pathImatge, String nomAudio, String codec, float durada, int kpbs): Semblant a afegirVideo, és el mètode encarregat d’afegir un Audio a la biblioteca. Creo l’objecte amb els atributs passats per paràmetre i l’afegeixo amb el mètode .add() a biblioteca, abans però, comprovant que existeixi la imatge en el disc. No es una opció que s’indiqui en la memòria però m’ha semblat oportú i útil ja que així sé si tinc un error perquè la imatge no es troba:



* 1. Void reproduirFitxer(int i): mètode que crida a la reproducció d’un fitxerReproduible de la biblioteca. Primer de tot, decremento aquesta posició, ja que jo vull que l’usuari triï a partir de la posició 1, i no zero. Després, controlo que la posició estigui dins del rang, és a dir, que no sigui superior al nombre d’elements afegits a la biblioteca, ni inferior. Si passa la comprovació, crearé un FitxerReproduible amb la posició passada, per poder cridar el mètode .reproduir().
  2. int indexAlbum(String titul): aquest mètode no es demana, però l’implemento jo perquè m’és molt útil. És un mètode que em retorna la posició d’un àlbum dins l’Array mitjançant el seu títol, passat per paràmetre. També em pot servir com a mètode equals(), ja que si no troba l’Àlbum que estic buscant, retorna -1.



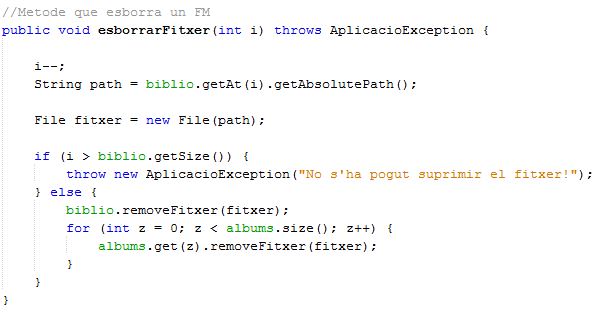
* 1. List<String> mostrarBiblioteca(): mètode que em mostra tots els elements afegits a la biblioteca, és a dir, és com un toString. Retorna un List de Strings, així que creo un ArrayList anomenat “aRetornar”, que contindrà tots els toStrings de cada fitxer afegit. Per fer-ho, recorro tota la biblioteca amb un for() i per cada element agafo la seva informació del propi mètode toString() de fitxer en qüestió i l’afegeixo al Array, el qual retorno.
  2. List<String> mostrarAlbum(String string): aquest mètode retorna una llista amb la informació completa d’un album afegit. La implementació és exactament la mateixa que mostrarBiblioteca(), nomes amb la diferencia de que obtinc l’índex de l’album mitjançant al mencionat mèotde indexAlbum(String titul) del títol passat per paràmetre.
  3. List<String> mostrarLlistatAlbums(): aquest mètode no és igual que l’anterior, ja que aquest retorna una llista amb el títol i la mida dels àlbums afegits al Array, no la seva informació. Això sí, també creo un nou ArrayList anomenat “aRetornar” que emmagatzema gracies a un for que recorre tot l’Array el toString() de cada album.
  4. Void afegirAlbum(String titol): aquest mètode afegeix un album al Array d’albums, amb un títol passat per paràmetre. Com ja em vist, el constructor d’àlbum se li passa un títol per paràmetre, que serà aquest mateix. Per tant, per afegir un àlbum, comprovo si el títol passat ja ha estat assignat a un altre album, amb el mètode indexAlbum() ja explicat. Si passa la condició, crido el mètode .add amb l’àlbum creat. Si no, llenço excepció.
  5. Void esborrarAlbum(String títol): com el seu indica, aquest mètode esborra un àlbum del Array. Per fer-ho, creo un enter que m’indicarà la posició on està afegit, amb el mètode indexAlbum(), per variar. Si aquest enter no és -1, és a dir, si he trobat l’àlbum, cridaré al mètode remove() d’ArrayList amb la posició del àlbum. Si no, llençaré una excepció.
  6. Void canviarTitolAlbum(String títol, String nouTitol): aquest mètode, dins de gestió d’albums, canvia el títol d’un àlbum, classe de la qual sabem que té un mètode setTitol() que rep un String per paràmetre. He fet que aquest mètode comprovi els dos títols passats; és a dir, que el títol “vell” existeixi i que el títol nou no existeixi. Ho faig amb dos enters, els quals se’ls hi assignarà el valor del títol passat al mètode indexAlbum(). Si passa la condició, accedeixo al Array d’Albums, concretament al àlbums que vull (Amb el mètode indexAlbum) i un cop el tinc, el mètode ja mencionat .setTitol(), amb el String de titolNou que m’han enviat desde AplicacioUB. Si alguna cosa no ha anat bé, llenço excepció.



* 1. void canviarMidaAlbum(String titol, int i): aquest mètode canvia la mida d’un Àlbum, classe de la qual sabem que té un mètode setMida(), que rep un enter per paràmetre. Per fer-ho, comprovo que el títol del àlbum ha canviar la mida que m’han passat existeixi, amb el mètode indexAlbum. Si aquest mètode no em retorna -1 (és a dir, que he trobat l’album) entro en el “if” i dins del Array trobo el àlbum en qüestió i li assigno la mida que m’han passat per paràmetre. Si alguna cosa no ha anat bé, com per exemple que no trobi l’àlbum, es llança una excepció.
  2. Void guardarDadesDisc(String camí): aquest mètode no es modifica de cap manera en aquesta entrega! En el meu cas, l’he mogut de AplicacióUB a dades. Com ja sabem, guarda les dades afegides a un fitxer en una ruta passada per paràmetre, gracies als FileOutput i ObjectOutputStream.
  3. Static Dades carregarDadesDisc(String camí): aquest mètode té la mateixa funció que en l’entrega 1, però es modifica en aquesta segona part. Com que el mètode readObject retorna un fitxer de tipus Object, abans fèiem un cast de FitxerMultimedia, però ara estem a la classe Dades, i ja no ho podem fer. La solució, doncs, és convertir el mètode en mètode classe, fent que sigui static. El que retornarem, en aquest cas, és un atribut de tipus Dades, que és el que es rep fent dades = (Dades) ois.readObject():



* 1. void afegirFitxer(String títol, int i): aquest mètode afegeix un fitxer que conté la biblioteca en un album. El primer que faig es decrementar la posició del fitxer en la biblioteca, perquè l’usuari ha triat un fitxer d’una llista que va del 1 a x però en veritat el primer fitxer correspon a la posició zero. Tot seguit, trobo l’album on afegiré el fitxer amb el mètode indexAlbum, de manera que, si aquest no és -1 (és a dir, existeix) entraré dins l’Array d’Albums, trobaré l’Album en qüestió i afegiré en ell el fitxer que es troba dins de biblioteca en la posició (getAt()) i, passada per paràmetre.
  2. void esborrarFitxer(int i): aquest mètode elimina un fitxer tant de biblioteca com de Album. Per fer-ho, l’usuari envia una posició corresponent a l’ubicació del fitxer en la biblioteca que servirà per cridar al mètode remove de biblioteca. És a dir, creo un fitxer amb el camí on es troba (getAbsolutePath()) el qual es necessari per fer biblio.removeFitxer(<fitxernou). Amb àlbums, però, és més complicat, ja que primer l’hagi de recórrer. El mètode és el mateix:



1. Preguntes

Les preguntes 1, 2 i 3 ja han estat respostes durant l’explicació de les classes i el desenvolupament.

* + - 1. *Expliqueu quants objectes s’han creat en l’execució d’aquest mètode main si estem a l’última línia;*

S’ha creat un objecte aplicació, de tipus AplicacioUB2, i seguidament al cridar aplicació.gestioAplicacioUB(); estic cridant al mètode que inicia tota la gestió del menú, amb creacions de mètodes Scanner, per exemple, la creació d’Objectes Menú, corresponents als submenús del Reproductor, ArrayLists... Depenent de l’ús que faci l’usuari del Reproductor, es cridarà a funcions i classes que a la vegada crearan nous objectes.

Si la pregunta només feia referencia en la classe main, només se’n crea un, aplicació.

* + - 1. *Expliqueu com heu implementat i on heu utilitzat el mètode equals heretat de la classe Object.*

Utilitzo equals() en el meu mètode indexAlbum(), per comparar el títol pasat amb el títol de cada album afegit en l’Array d’Albums. A partir d’allà, sempre faig servir aquest mètode i no “existeixAlbum(String títol), ho he preferit així.

* + - 1. *Les preguntes 6 i 7 ja han estat respostes en l’explicació de les classes i el desenvolupament.*
      2. *Expliqueu si heu fet servir la sobrecàrrega de mètodes a la classe Controlador i detalleu com i perquè.*

He fet sobrecarrega en tots els mètodes (menys el constructor) de Controlador perquè tots deriven de la classe Dades, és a dir, Controlador controla l’accès a les dades, per tant, cada mètode de Dades ha de ser supervisat, amb @Override, per Controlador.

* + - 1. *Completeu el diagrama de classes proporcionat en l’ajut del lliurament (Punt 4 del document) per tal de mostrar el recorregut que fa el vostre programa quan s’executa l’opció d’afegir un fitxer multimèdia a la biblioteca*

2.1

1

2

Existeix i no està afegit?

FitxerMultimedia

Llença AplicacioException

No?

2.22

Si?

Afegeix-lo a la taula de Fitxers

* + - 1. *Expliqueu les proves realitzades per comprovar el correcte funcionament de la pràctica, resultats obtinguts i accions derivades.*

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

* + - * + M’afegeix i elimina fitxers tant a Album com a Biblioteca, amb els atributs i les condicions indicades
        + Em reprodueix qualsevol Video o Audio afegit
        + Guarda i recupera dades d’un fitxer
        + És capaç de modificar la mida i el títol d’un Album
        + És capaç d’imprimir per pantalla la informació d’un o més d’un fitxer
        + És capaç de gestionar diferents excepcions que es puguin donar en qualsevol de les opcions del Reproductor per tal de no para l’execució d’ell.

Per altra banda, m’agradaria afegir que s’agraeix que l’entrega s’hagi ampliat ja que realment el salt entre una entrega i l’altre és molt gran, sobretot perquè “sobrepasses” la dimensió de Java amb sí en entrar en contacte amb un altre programa com VLC Player. A més, és el primer contacte amb classes Abstractes i amb un Controlador.