Memòria #3

Introducció

Seguidament del 1, faig un incís dels problemes i les solucions aplicades al desenvolupament d'aquesta pràctica. Ho faig després del punt 1 i no a observacions perquè crec que és bo conèixer-los abans de passar a punts posteriors. Això es pot considerar com els apartats introducció i desenvolupament, però en aquesta entrega veig més convenient fer-ho seguit del punt 1.

1. Introduïu el problema tractat en el lliurament (No s'acceptarà una còpia directa de l'enunciat).

1. Es demana poder visualitzar les imatges de diverses maneres:

- Sense filtre
- Amb filtres sípia
- Amb filtre en blanc i negre
-

2. Es demana poder transformar les imatges de diverses maneres:

- Amb filtre sípia
- Amb filtre blanc i negre
-

3. Es demana que una imatge filtrada pugui tornar al seu estat original (treure els filtres aplicats):

- Per a poder visualitzar una imatge cal que aquesta existeixi al disc. Per a fer-ho crearem primer el fitxer a partir del mètode proporcionat saveImage() que heredem <ImageFile>.
- Comprovarem si el fitxer ja ha estat creat abans d'utilitzar aquest mètode ja que així estalviarem el cost computacional de la funció, en cas que ja existeixi.
- En el meu cas per visualitzar/transformar imatges em limito a enllaçar i desenllaçar els objs a la url que toca per apuntar al fitxer de disc desitjat.

El que faig és tenir un mètode per posar flitres i un mètode per treure filtres a una imatge filtrada.

4. Es demana poder visualitzar la biblioteca o un àlbum utilitzant el mètode proporcionat onTimer() a la superclasse <ImageFile>:

- En el meu cas es visualitza una llista d'imatges (la que es demani) i quan acaba torna a començar per la primera imatge. Hi ha una opció al menú per aturar la visualització. És l'opció (8).

Problemes i solucions d'aquest lliurament

Seguint el que es va dir a classe, quan es transforma una imatge de la biblioteca, només queda transformada a la biblioteca tot i poder estar afegida a un o més d'un àlbum, una vegada o moltes vegades.

Això porta una sèrie de problemes alhora d'eliminar imatges ja que si per exemple obrim el visor, afegim una imatge a biblioteca, tot seguit creem 3 albums, afegim aquesta imatge de la biblioteca a tots els àlbums 5 vegades, llavors tornem a la biblioteca i transformem la imatge a Sepia. Ara si eliminem la imatge de la biblioteca quedarà la biblioteca buida i 15 imatges (5 a cada àlbum) ja que el mètode equals ja no les considera iguals perquè ara li hem canviat el path. Això no és possible ni està ben implementat ja que no podem tenir la biblioteca buida i àlbums plens (seria com tenir un reproductor de música buit però amb llistes de reproducció plenes de cançons).

Per solucionar aquest problema he programat el visor perquè, quan volem eliminar una imatge de la biblioteca, elimini totes les imatges dels àlbums associades a una imatge original tot i ser transformades tant a la biblioteca com en els àlbums. L'eliminació d'una imatge en un àlbum segueix igual que abans ja que no ens afecta.

Els casos són els següents:

- Cas 1 : Afegim una imatge a la biblioteca, la afegim a un àlbum i sense transformar-ne cap eliminem la de la biblioteca → Resultat : S'eliminen totes dues
- Cas 2 : Afegim una imatge a la biblioteca, la afegim a un àlbum, transformem la del àlbum i eliminem la de la biblioteca → Resultat : S'eliminen totes dues
- Cas 3 : Afegim una imatge a la biblioteca, la afegim a un àlbum, transformem la de la biblioteca i eliminem la de la biblioteca → Resultat : S'eliminen totes dues
- Cas 4 : Afegim una imatge a la biblioteca, la afegim a dos àlbums, en un àlbum la transformem a BN, a l'altre àlbum no la transformem, i a la biblioteca la transformem a Sepia. Després eliminem la de la biblioteca. → Resultat : S'eliminen les tres
- I així per la resta de casos...
- Les portades també canvien sempre que cal.

Com he implementat això?

Per a fer-ho em serveixo d'un mètode removeFilters() que em retorna la imatge amb el path original, és a dir, li treu el filtre que tingui. Llavors al comprar la imatge per a ser eliminada la comparo obviant el filtre que tingui. Així va perfecte.

La primera idea que vaig tenir va ser retocar el mètode equals i fer que comparés el path traient-li el filtre directament allà, però el problema era que si primer afegia una imatge.jpj amb nom "pau" i després volia afegir una imatge_BN.jpg que ja tenia creada al disc dur amb nom "pau" em deia que ja existia quan en realitat no era així.

Llavors la solució a això és comparar fora del mètode equals i a més aquesta comparació es fa just on es feia en l'anterior entrega quan volíem eliminar totes les imatges dels àlbums que coincidissin amb una imatge que eliminem de la biblioteca. Però ara a la comparació li afegim el obj.removeFilters() i queda solucionat.

Pero encara no és suficient...

El problema ara és el següent:

Imaginem que amb al pc hi tenim una imatge pau.jpg i una imatge pau_SP.jpg, aquesta última la tenim perquè anteriorment hem utilitzat el visor, per exemple, i ja està filtrada a sèpia. Llavors ara nosaltres creem un visor nou i afegim totes dues imatges a la biblioteca i ho fem amb el mateix nom "holaa", de manera que a la biblioteca hi tenim:

- Nom: holaa, path: pau.jpg
- Nom: holaa, path: pau_SP.jpg

Tot seguit afegim totes dues imatges a un àlbum i llavors anem a la biblioteca i eliminem la primera imatge. El més normal i el que volem és que ara a la biblioteca quedi una imatge (doncs n'hem eliminat una) i a l'àlbum també hi quedi una imatge. ¡Però no és així! Doncs a l'àlbum s'han eliminat les dues imatges degut a que la comparació es fa amb el path 'desfiltrat' i el nom i aleshores el programa les detecta iguals i les borra les dues de l'àlbum.

La solució...

La solució és crear un atribut ID a la classe Imatge i ara cada imatge portarà associat un enter que serà un ID i que no es demanarà a l'usuari sinó que s'anirà auto incrementant sol amb cada creació de nova imatge, l'usuari ni tant sols sabrà que existeix. El comptador d'aquest ID no podrà ser un atribut static de la classe imatge que ens compti les instàncies ja que durant el programa es creen molts objs <Imatge> temporals i no funciona perquè es perd el compte i fallen les comparacions.

Aquest atribut l'he col·locat a <DadesVisor> com a privat i es va incrementant cada vegada que s'afegeix una Imatge a la biblioteca assignant un ID únic a cada <Imatge>.

La comparació per ID només es fa en un sol lloc del programa i aquest lloc és l'Iterador que ens vareu demanar fer a la pràctica anterior perquè quan eliminéssim una imatge de la biblioteca recorres tots els àlbums eliminant-ne les ocurrències. Ara el programa només considera imatges iguals a efectes d'eliminació aquelles que a més de tenir el nom igual i el path (sense filtre) igual coincideixen per ID.

No he tocat el mètode equals que es queda com està comparant path i nom de la mateixa manera que sempre.

La conclusió que trec de tot això és que hem de poder afegir a la biblioteca dues imatges que tot i tenir el mateix nom tinguin paths diferents com pau.jpg i pau_BN.jpg ja que són arxius físics diferents i no són en cap cas el mateix fitxer en versions diferents. El sistema operatiu pot haver guardat pau.jpg a la posició @1998H de la memòria i l'altre pot estar a la posició @0034H de la memòria. SON DIFERENTS.

El problema d'eliminar imatges del PC i recuperar dades de disc amb el visor...

Si l'usuari decideix un dia fer neteja del PC i elimina imatges que tenia guardades en un fitxer de dades que pertany al visor, tenim un problema. És que l'usuari el dia que vulgui recuperar les dades del visor, per exemple 'dades.dat', aleshores quan visualitzi imatges que ja no estàn a disc el programa petarà.

Això és un aspecte que s'ha de controlar i fer alguna cosa perquè això no passi...

Per solucionar això cada vegada que recuperem dades de disc, just després de ser recuperades, cridem al mètode updateVisor() de la classe <DadesVisor>.

El primer que fa aquest mètode és recórrer la biblioteca i eliminar totes les imatges que contingui la biblioteca però que han deixat d'existir a disc. Aquest primer pas també recorre tots els àlbums per eliminar totes les ocurrències d'aquesta imatge. (Té en compte els possibles canvis en la portada dels àlbums).

El segon pas, just a continuació, ha de tornar a recórrer els àlbums per mirar si en algun àlbum hi ha una imatge que ha deixat d'existir a disc però que no pertany a la biblioteca. Aquest és el típic cas en el qual hem aplicat un filtre a la imatge d'un àlbum però no a la biblioteca. Aquesta només explorant la biblioteca clarament no serà detectada. Aquest segon pas també té la virtut d'actualitzar si cal, automàticament, la forma foto de portada.

Una altra "solució" podria ser pensar que si una imatge no existeix a disc quan la visualitzem fem retornar una excepció, però el que passa és que trobo força cutre que estiguis visualitzant un àlbum amb l'onTimer i vagin apareixent excepcions a la consola, bé realment no sé si passaria així ja que ho veig una solució tant cutre que ni la he provat de fer.

El problema de les repeticions d'imatges quan apliquem filtres...

Un dels altres problemes és que si per exemple tenim a la biblioteca pau.jpg i pau_BN.jpg ara impedeixo que pau.jpg es pugui transformar a pau_BN.jpg sempre i quan tinguin el mateix nom (l'equals torni true). Si no fem això es pot donar el cas de tenir dues imatges iguals a la biblioteca tot i que no sigui afegint una imatge, es a dir, ho hem de frenar abans que passi.

Per a fer-ho he creat el mètode replaceObj a la classe <llistaImatges> cridat desde <DadesVisor> i li he incorporat una excepció tipus VisorException amb el seu pertinent missatge de sortida en cas que la imatge ja existeixi i a més, només si es tracta de la biblioteca ja que en els àlbums si que podem tenir imatges repetides.

Un aspecte que no m'he atrevit a canviar per no sortir-me del guió del visor...

Manipulant el visor a cada entrega, cada vegada penso més sovint a treure l'arraylist de la llista dels àlbums de <DadesVisor> i fer una classe a part que es digui <LlistaAlbums>, cosa que crec que té molta més lògica, però no ho faig perquè no se si ho veuries bé i passo d'anar regalant punts de la nota.

El filtratge i la imatge de portada ...

Un aspecte a tenir en compte quan filtrem una imatge d'un àlbum és que si la imatge que filtrem és la portada, la portada s'haurà de canviar per la imatge que hem filtrat.

2. Expliqueu les classes implementades.

2.1 - Classes ja implementades en pràctiques anteriors

<Imatge> : Representa un obj Imatge i hereda de <ImageFile> que a la vegada hereda de File.

<Llistaimatges> : És la classe base per gestionar les llistes d'imatges, hereda de la classe abstracta <ImageList> i ens imposa el diseny de l'interface <InImageList>.

<AlbumImatges> : Un obj AlbumImatges i és fill descendent de <LlistaImatges> ja que un àlbum és una llista d'imatges.

<BibliotecaImatges> : És filla descendent de <LlistaImatges>, fa servir algunes de les funcions "mare" i sobreescriu les que són necessaries tot i fent ús del polimorfismse.

<DadesVisor> : És la classe on es crea l'array dels àlbums (jo l'hagués creat en una classe a part però un disseny imposat és un disseny imposat). Aquesta classe, a més, conté totes les dades del visor. Des d'aqui, a més, és d'es d'on guardem i recuperem les dades ja que l'objecte serialitzable que guardem/recuperem és precisament un obj <DadesVisor>. Aquesta classe doncs és la que portarà implements Serializable.

<CtrlVisor> : Exerceix d'intermediari entre el model i la vista. Crea un obj <DadesVisor> i el fa servir per relacionar-se amb les llistes d'imatges i la llista dels àlbums. Aquesta classe és cridada a través d'un obj per la vista.

<VisorUB2> : Classe que gestiona l'aplicació i que conté els menús. Crea un obj del controlador per poder interactuar indirectament amb el model.

<GestioVisorUB> : conté el main. Crea l'obj Scanner del qual es beneficia tota l'aplicació. Crea un obj <VisorUB2> a través del qual crida al mètode gestioVisorUB() de la classe <VisorUB2> per posar en marxa l'aplicació.

2.2 - Classes ja implementades per al desenvolupament del visor 3

<ImatgeSepia>: És filla directa d'Imatge. Hereda tots els mètodes d'Imatge i té creats 3 mètodes propis que serviran per gestionar la classe:

- color2Sepia() → Mètode que aplica un filtre sípia sobre la imatge

- save() → Guarda la imatge utilitzant el mètode saveImage() de la classe <ImageFile>. En aquesta classe és un mètode sobreescrit ja que també l'he implementat a Imatge().
- Show() → sobreescriu el show proporcionat per la llibreria utils que ens heu fet implementar. Aquest mètode es limita a cridar al'anterior (save) i després crida al show() de la classe mare <Imatge>, aquest és qui dona tamanys genèrics a la imatge i després crida al show() d'<ImageFile> que ja ensenya la imatge.

<ImatgeBN>: És exactament idèntica que la classe <ImatgeSepia>.

<Filtrable> ***Interface*** : És una interfície creada amb tots els mètodes necessaris per filtrar una imatge o visualitzar-la amb filtre. És un patró imposat per mi el qual han de seguir tots els elements que siguin filtrables. En aquest cas imatges. L'explico a continuació:

Interface <Filtrable>

Com s'implementa aquesta interface?

La interface <Filtrable> ha estat creada per donar una forma comú a tots els elements del visor que seran filtrables: les imatges. A continuació explico els motius del "com" i el "perquè" de la meva implementació.

Aquesta interface la implementa la classe <Imatge> degut a que una imatge és aquell element que pot ser filtrat. El benefici d'això és que ara totes les classes filles: <ImatgeBN>, <ImatgeSepia> i totes les que hi pugui haver ja tenen aquests mètodes implementats perquè els hereten d'<Imatge>, la seva superclasse directa.

Una altra opció hagués estat fer que fossin les classes filtrades les que implementessin la interface <Filtrable>, però el **problema** d'això és que llavors totes aquestes s'haguessin vist obligades a implementar tots els mètodes. Ara no hi ha gaire diferència inclús es podria considerar que són aquestes classes les que haurien d'implementar aquesta interface, però si pensem en gran la cosa canvia. Demostració:

- Ara tenim 2 classes amb filtre i la interface obliga a implementar 5 mètodes. Si ho féssim així tindríem un total de 10 mètodes. No passa res. Només tindríem 6 mètodes inútils més que si ens fixem en la manera com ho he implementat jo.
- Però si tinguéssim un visor amb 200 filtres tindríem 1000 mètodes implementats ocupant un espai inútil. **"Només"** 995 mètodes més que utilitzant la meva implementació, la qual força només a implementar 5 mètodes a la classe <Imatge> independentment de quants fills tingui aquesta. Lògicament aquests fills es beneficien igualment dels 5 mètodes.

Però aquest no seria l'únic problema...

Tot l'animalada de mètodes extra descrits en l'apartat anterior, aquest potser no seria el problema més greu. Imagina't que en el moment en què tenim 200 classes (1000 mètodes implementats obligatoris, 5 a cada classe) ens demanen modificar el cos d'un o dos d'aquests mètodes perquè a més del que fan, ara facin alguna cosa diferent. Hauríem de modificar 1000 mètode obrint 200 fitxers .java i modificant el codi... brutal!!

Suposem ara que per cada fitxer (anant despresa) triguem 1 minut en obrir-lo, modificar els mètodes, guardar-lo i finalment, tancar-lo. Trigaríem 200 minuts!! 3 hores i 20 minuts modificant mètodes. I suposant que no fem cap descans! Casi res!

I això es podria complicar molt més si la interface enlloc de tenir 5 mètodes en tingués 15! O 20! O més!

Aquestes qüestions són el motiu pel qual no faig implementar la interface de filtratge sobre les classes <ImatgeSepia> i <ImatgeBN>. Per això veig molt més intel·ligent fer que directament la classe <Imatge> implementi la intercafe. Ens trobem davant d'un cas de -Disseny estricte Vs Disseny intel·ligent -, jo trio l'intel·ligent.

3. Expliqueu quines classes has pogut reutilitzar del primer lliurament per a fer aquest. Quins canvis sobre les classes reutilitzades heu necessitat fer i perquè.

Totes les classes de la pràctica anterior han estat lògicament reutilitzades i continuen fent servei.

Algunes de les modificacions de classes així com implementacions fetes:

Com es visualitza una imatge?

A <VisorUB3> hi ha un menú (menuVisualizeImage) que té tantes opcions com filtres existeixen + una opció per visualitzar sense filtre. Cada opció del menú crida a un mètode de suport independent de visualització: showImage(), showImageSepia() ... etc...

Aquests mètodes de suport reben un entre que és el número d'àlbum del qual volem visualitzar una imatge, si rep un -1 és que volem visualitzar directament de la biblioteca. Així doncs la primera conclusió que podem treure és que cada filtre tindrà els seus propis mètodes, però que aquest mètode és perfectament capaç de ser compartit tant per àlbums com per biblioteca ja que les dues coses son llistes d'imatges i ens val enviar-li un -1 perquè el programa sàpiga que ha d'anar a buscar la imatge ala biblioteca i no a un àlbum.

Dit això, aquests mètodes show() de suport criden a un mètode getPos() de suport que deixa triar una imatge a l'usuari i ens retorna la seva posició. Això ho faig en un mètode a part degut

a que és codi que es repeteix a molts mètodes i així faig el meu programa més modular, robust i compacte.

Una vegada tenim la posició (imatge) i la list (llista que volem utilitzar) cridem al showImage() del controlador, o showImageSepia() o el que sigui enviant-li aquests dos arguments. Llavors en aquest mètode i utilitzant aquests dos paràmetres creo l'obj <Imatge> que vull visualitzar i crido al mètode de <dadesVisor> passan-li l'obj, aquest mètode ja és totalment modular i li és igual si li passes una imatge filtrada o no, ja ensenyarà la que toca igualment cridant al show del propi obj.

Recordem que la classe imatge ja la tenim perfectament preparada amb una interface que té la capacitat de filtrar qualsevol imatge i ensenyar-la.

Com es transforma una imatge?

El procediment inicial a <VisorUB3> és exactament el mateix, és a dir la llista i la posició es cacen de la mateixa manera i es crida al controlador exactament igual, és a dir, enviant-li un índex de llista i un índex de posició dins aquesta llista.

El controlador crea l'obj <Imatge> de la mateixa manera però ara a més es crea un obj <LlistaImatges> que és la llista sobre la qual volem fer el replace d'un obj "normal" per un obj al qual hem afegit un filtre a la seva url. Aquest mètode crida a un mètode de <DadesVisor> que pot rebre tant imatges normals com imatges filtrades ja que rep un paràmetre <Imatge>. És des d'aqui a dades visor d'es d'on es crida a guardar la imatge a disc classe <Imatge> i d'es don es crida al mètode de <LlistaImatges> que farà la substitució en el seu propi array.

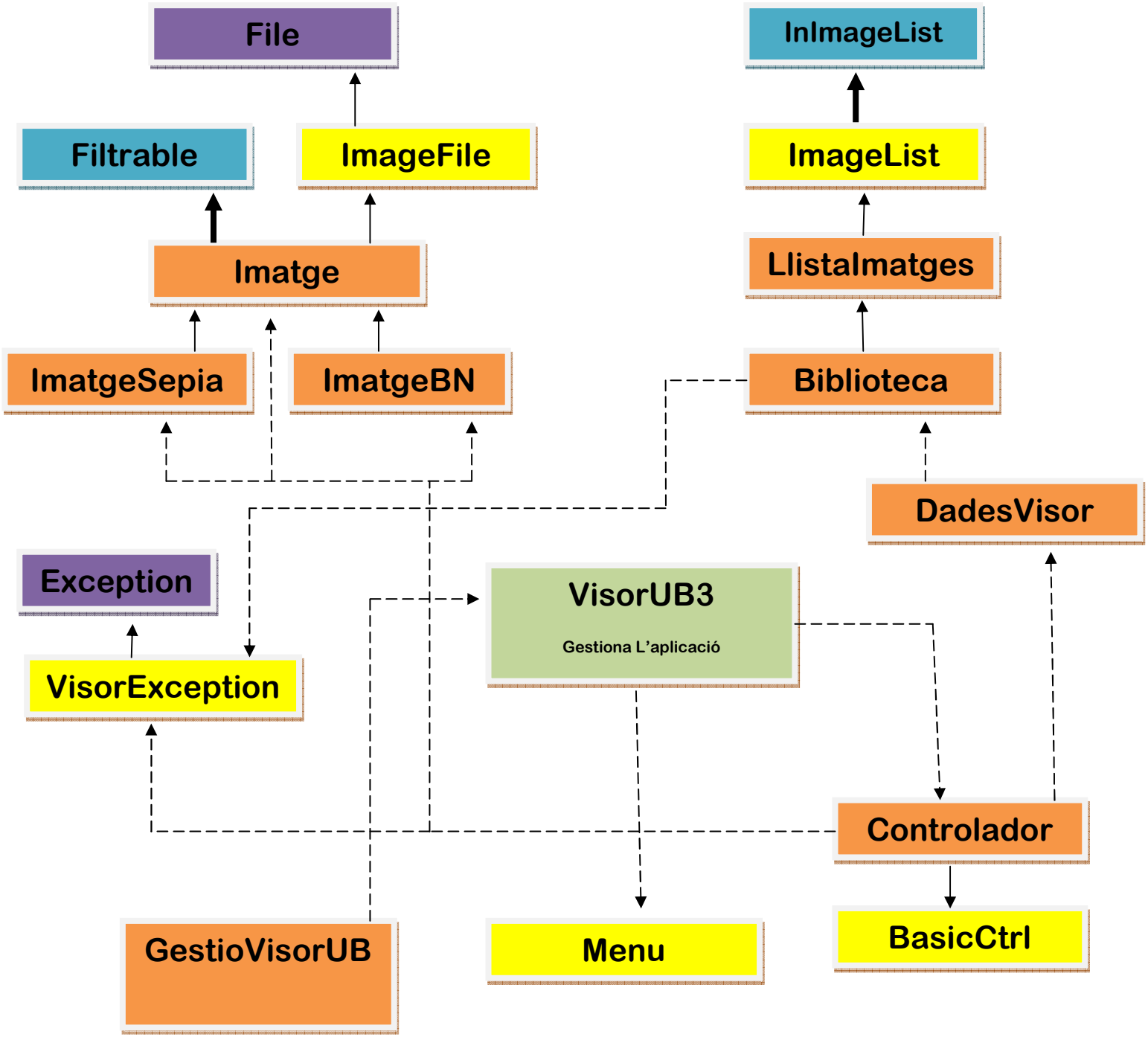
Si la llista és un àlbum i la imatge transformada era la imatge de portada, es fan les accions necessaries per actualitzar la portada del àlbum.

Pregunta 4:

L'esquema està a la següent pàgina perquè aqui no hi cap!

4. Dibuixeu el diagrama de relacions entre les classes que heu utilitzat a la pràctica. Incloure tant les classes implementades per vosaltres com les que pertanyen a la llibreria UtilsProg2.

Lila → Classe de java
Groc → Classe d'utilsProg2
Blau → Interface
Linia gorda → Interface
Linia prima → herencia
Linia disc. → instanciació



Falta la interface Serializable (no m'hi cap), però la tinc en compte!!

5. Fes un esquema mostrant el recorregut que fa el teu programa quan s'executa l'opció de visualitzar imatges de la biblioteca. Especifica els mètodes que es criden en cadascuna de les classes. Fes servir números per indicar l'ordre de les criden.

Entenc que **visualitzar imatges (així escrit en plural tal i com diu l'enunciat)** es refereix a visualitzar una llista d'imatges amb l'onTimer() perquè sinó estaria escrit en singular. De tota manera com que no queda clar faré els esquemes tant de visualitzar amb l'onTiner() com de visualitzar una imatge de manera individual. Ara bé explicar amb text només t'explico l'onTimer ja que la visualització individual te la he explicat en un dels apartats anteriors amb tota mena de detall.

La visualització d'un àlbum o la visualització de la biblioteca és el mateix des del sentit que les dues coses són llistes d'imatges i com a tals es tracten de manera igual. Tot comença al menú de la biblioteca o al menú d'un àlbum indistintament. Desde l'ació del menú es crida al mètodes de suport slideImages(Object list) que rep una llista com a paràmetre de tipus Object ja que a la vista no podem fer imports del model. Aquest mètode crida al mètode play() que es troba al controlador. El mètode play en el controlador rep la llista encara de tipus Object i la casteja a <LlistaImatges> per a fer-la servir com a tal. I aquí faig un incís per explicar perquè ja creo un obj de tipus ImageDialog aqui dins i no m'espero a fer-ho en el onTimer():

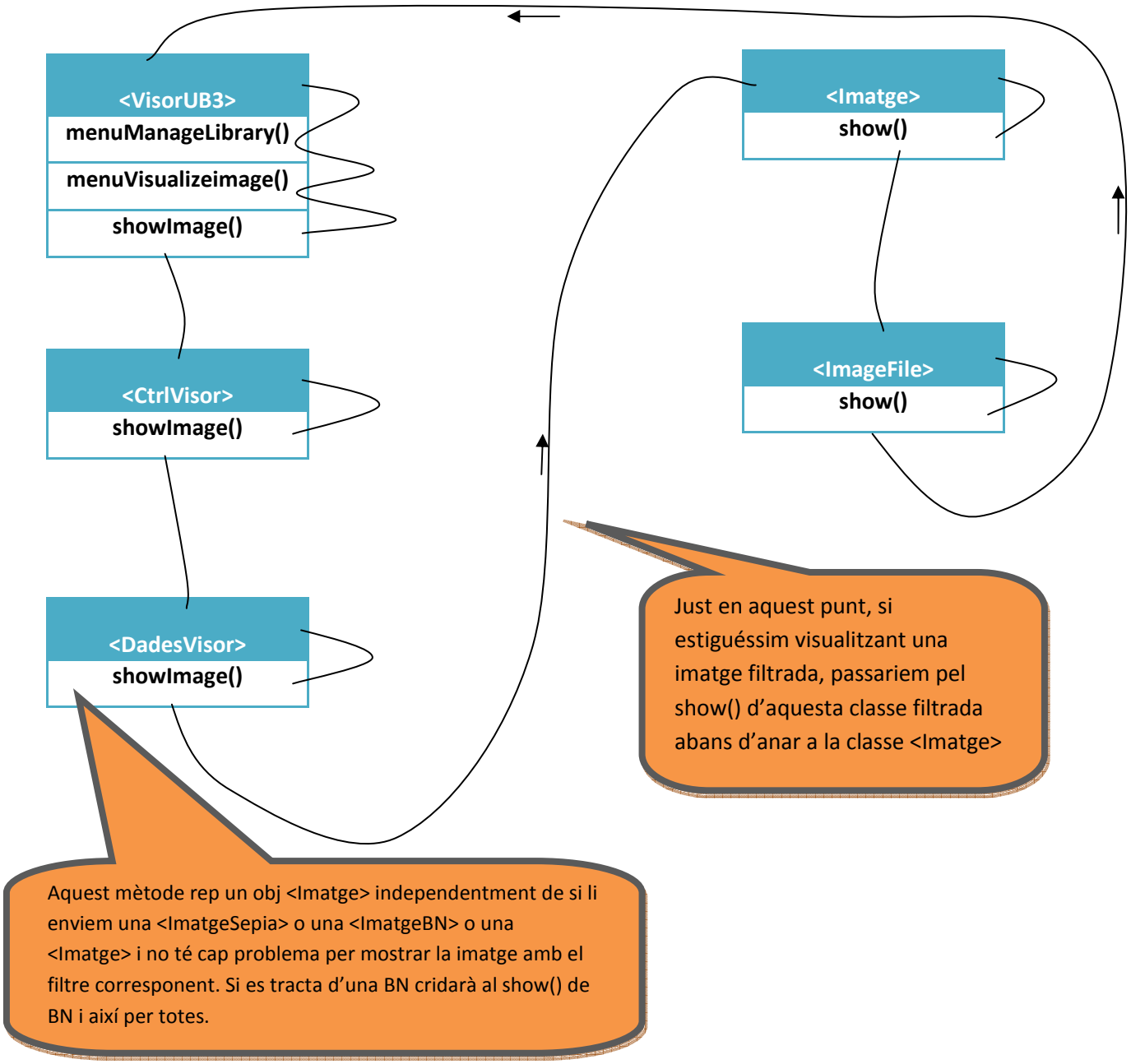
- Jo tinc un temps entre imatge i imatge de 4 segons. L'onTimer() s'espera llavors a que passin 4 segons per ensenyar la primera imatge. Per evitar això jo ja ensenyo la primera imatge en el mètode play(), incremento el comptador i llavors ja crido a l'onTimer() que seguirà a partir de la segona imatge. Així s'evita un temps d'espera que pot portar a confusions si no podem un missatge tipus "loading..."

Fet això i encara dins el mètode play() primer activo el setTimer(4000) a 4 segons i tot seguit crido al mètode onTimer() que ensenyarà totes les imatges i després tornarà a començar fins que l'usuari pari la visualització amb l'opció pertinent del menú.

Per a fer tot això al controlador he preparat dues variables globals currentImage i currentList que es posen automàticament a zero quan aturem la visualització i queden a l'espera de tornar a a ser utilitzades de nou.

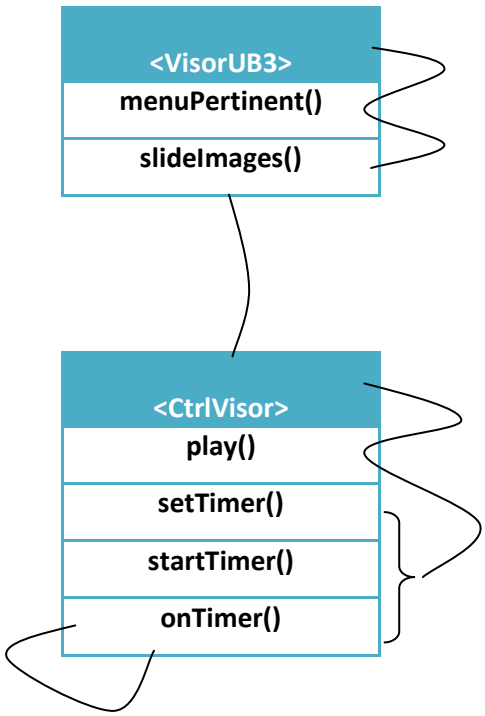
Els esquemes estan a les pàgines següents perquè aquí no hi caben!

1. Visualització d'imatges de forma individual.

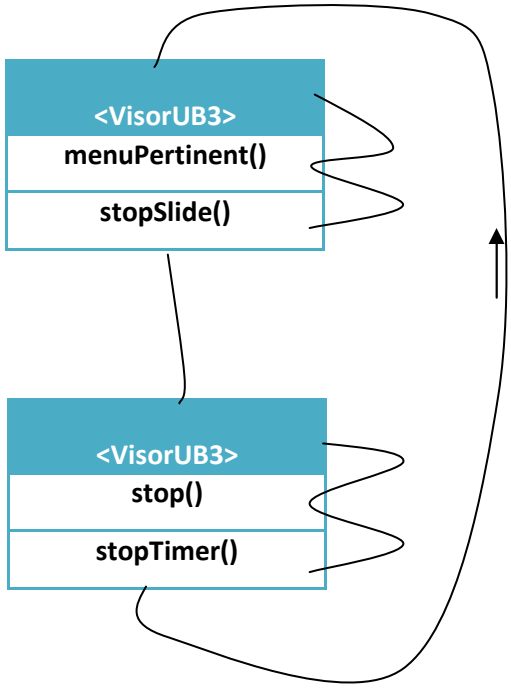


2. Visualització d'imatges amb l'onTimer.

Activar la visualització



Desactivar visualització



10. Expliqueu les proves realitzades per comprovar el correcte funcionament de la pràctica, resultats obtinguts i accions derivades.

En computadors poc potents com el de casa meva he tingut i segueixo tenint un problema amb la visualització de les imatges amb l'onTimer():

- Problema: em computadores poc potents, quan ja hem visualitzat algunes imatges, el programa omple la memòria heap i el programa peta. ¡Alerta! Això només em passa a casa meva, en els pc's de la uni sembla que són més potents i funciona bé. He provat

de posar una imatge a null després de ser visualitzada i tot seguit forçar el system.gc() per recollir els objs des-referenciats però no funciona. Et deixo l'error que apareix i en podem parlar si vols a classe de pràctiques. L'error és el següent:

Exception in thread "AWT-EventQueue-0" java.lang.OutOfMemoryError: Java heap space at java.awt.image.DataBufferByte.<init>(DataBufferByte.java:92)

De tota manera ja et dic que si la teva computadora és prou potent no veuràs l'error perquè no es produirà.

11. Observacions generals.

Les observacions generals les he anat comentant durant aquest escrit quan ho he anat necessitant, així que ja no sé què més puc dir. Aprofito aquest apartat per dir-te algunes coses sobre la retroacció anterior.

Tal i com demanaves a la retroacció anterior el mètode removeImage() de la classe BibliotecaImatges, que no calia, ha estat eliminat tal i com vas suggerir.

Tal i com demanaves a la retroacció anterior he arreglat els temes de la portada, que realment no era un error sinó que jo ho implementava diferent, ara bé, ja està tot com es demana pq ho he canviat.

Tal i com demanaves a la retroacció anterior ja no crido al mètode addImage() de la classe llistaimatges des del Controlador sinó que ara passo per la classe <dadesVisor> que és qui conté totes les dades del nostre visor.

Durant aquesta memòria he utilitzat frases i expressions bastant dramàtiques perquè s'entengui bé el que he fet, com ho he fet, per quin motiu ho he fet i quines són les ventatges d'haver-ho fet així..