Memòria #2

Introducció

L'objectiu de la pràctica és seguir evolucionant el nostre visor afegint noves classes i un nou mòdul que serà el controlador perquè la vista i el model no tinguin contacte directe. A més introduirem les excepcions per tractar alguns errors. No m'enrollo més aquí perquè ho explicat tot a fons en els apartats posteriors.

Anàlisi i desenvolupament

Aquest és bon punt per explicar que tal i com em vas suggerir he canviat de tenir un fitxer de dades fixe a deixar a l'usuari que esculli on vol col·locar el fitxer de dades i on el vol guardar.

Aprofito també aquest punt per explicar-te que per tractar els fitxers he instal·lat un FileChooser gràfic per comoditat alhora de fer les proves i ja de pas l'he deixat posat .

El tema és que super-pesat estar fent proves i haver d'introduir les rutes a mà, per aquest motiu, he instal·lat un JFileChooser de Swing purament per comoditat (sóc conscient de que l'exercici no ho demana) però haver d'introduir rutes que de veritat existeixin al PC amb tot el path, nom de file i extensió és incòmode si has de repetir l'acció moltes vegades i fa perdre molt de temps. I per fer proves cal repetir aquesta acció moltíssimes vegades tant com per adjuntar imatges com per guardar i recuperar les dades.

Ja que he decidit fer-ho així ho he fet robust i ben instal·lat retornant tots els missatges d'error, comprovant les valideses de les extensions, etc. A més possiblement em serveixi per quan fem la part gràfica (tant de bo) o possiblement no.

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

Es demana ampliar el Visor d'imatges ara seguint el disseny model-vista-controlador on la vista no pot relacionar-se directament amb el model ni crear-ne cap objecte. Ambdós contacten a través del controlador que actua en els dos sentits per poder relacionar correctament el model i la vista.

Ara, a més tractarem la gestió d'alguns errors mitjançant excepcions. Aquestes es detectaran a la part del codi on correspon i es resoldran a la vista, lloc on mostrarem els missatges d'error.

La classe TaulaImatges ha deixat d'existir en benefici de <LlistaImatges>, la classe que encara resta útil a la pràctica.

Ara el menú serà un menú de sub-menús tots coordinats i relacionats l'un amb l'altre degudament.

Comencem a utilitzar gran part del potencial del polimorfisme après a les classes de teoria.

2. Expliqueu les classes implementades.

<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.

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.

Seguim utilitzant totes les classes excepte <TaulaImatges>. He pogut reaprofitar codi de la classe <LlistaImatges> en les seves dues subClasses <AlbumImatges> i <bibliotecaImatges> tot i aplicant el concepte de polimorfisme. Per altra banda també he creat classes noves esmentades a l'exercici anterior.

Per citar algún exemple → <BibliotecaImatges> sobreescriu els mètode addImage i removeImage però profita tots els altres de la seva superclasse sense necessitat de sobreescriptura. Llavors el programa detecta en temps d'execució a quin mètode ha de cridar segons el tipus d'obj que es tracti.

De la mateixa manera <AlbumImatges> només sobreescriu addImage ja que en un àlbum el removeImage m'interessa sobrecarregar-lo per poder eliminar a través de l'índex ja que pot tenir imatges repetides. Així doncs la capcelera del mètode a <LlistaImatges> és removeImage(ImageFile obj); mentre que a <AlbumImatges> és: removeImage(int pos);

Això vol dir que <AlbumImatges> es beneficia dels dos mètodes, el de la superclasse que rep una imatge i el this que rep un enter.

4. Expliqueu quants objectes s'han creat en l'execució d'aquest mètode main si estem a l'última línia:

```
public static void main(String[] args) {
        VisorUB2 vista=new VisorUB2();  // Creem un objecte de la vista
        vista.gestioVisor(); // Inicialitza l'execució de la vista
}
```

El primer obj que es crea és el que veiem <VisorUB2>. El costructor de <VisorUB2> crea un obj <CtrlVisor> i el seu constructor crea un obj <DadesVisor>, el costructor de <DadesVisor> crea dos objectes: un <BibliotecaImatges> i un <ArrayList> d'<AlbumImatges>. <BibliotecaImatges> crida al seu super <LlistaImatges> que en el seu constructor crea un obj <ArrayList> d'imatges i l'envia creat al seu super que és <ImageList> que costruiex l'array a través del seu setter.

Obs // Entremig de tot això jo (en el meu cas particular) a <VisorUB2> creo un obj JFileChooser per manipular fitxers amb més comoditat.

5. Expliqueu com heu implementat i on heu utilitzat el mètode equals heretat de la classe Object.

El mètode equals està sobreescrit a dues classes: <Imatge> i <AlbumImatges>.

A la classe imatge (tal I com diu l'enunciat), aquest mètode vetllarà per la igualtat de dos objectes <Imatge> quan coincideixen en el seu nom i la seva ruta. Si no és així, dues imatges no es consideraran iguals. És a dir: ruta + nom iguals → return true.

A la clase <AlbumImatges> el nostre mètode sobreescrit equals() compararà dos obj <AlbumImatges> per títol i ens dirà que dos àlbums són iguals si els dos tenen el mateix String per títol. És a dir: títol igual → return true.

6. Expliqueu com heu implementat la comprovació de que una imatge no està inclosa en la biblioteca.

Comprovo si una imatge està o no a la biblioteca mitjançant un mètode imageExists() que m'he creat a <BibliotecaImatges>. Aquest mètode utilitza el mètode contains() de <List> de java. Podia haver utilitzat contains() directament però queda més polit en una funció:

```
public boolean imageExists(Imatge obj){
    return getList().contains(obj);
}
```

Aquesta comprovació es fa tant per afegir una imatge com per eliminar una imatge.

En cas que estiguem afegint una imatge, la deixarem afegir sempre i quan aquest mètode retorni false. En cas contrari voldrà dir que ja existeix i llançarem l'excepció corresponent.

Cal posar èmfasi a que el mètode contains() es basa en el nostre mètode equals sobreescrit per decidir si l'array conté o no l'objecte. Si no l'haguéssim sobreescrit, llavors utilitzaria l'equals de sèrie (no e sobreescrit)

En cas que estiguem eliminant una imatge només serà possible quan la funció retorni true. Obs// Quan elimino una imatge no retorno excepció pq l'enunciat només parla de retornar excepcions quan afegim una imatge (i no quan eliminem).

De tota manera, quan volem eliminar una imatge el programa mostra tota la llista ordenada d'imatges amb el seu índex al davant. L'usuari entra per teclat un índex i si aquest no està en el rank del size() de la llista d'imatges llavors torna a demanar un numero a l'usuari fins que aquest es troba dins el rank o es cancel·la l'operació. Aquesta és la manera més eficaç de comprovar que una imatge evidentment existeix abans de ser eliminada. No obstant i per què ho diuen els enunciats, també ho comprovo amb la funció que he esmentat que conté el contains().

7. Expliqueu com heu utilitzat la classe VisorException al vostre codi.

La clase VisorException la utilitzo per retornar l'error de quan no és posible afegir una imatge a la biblioteca ja sigui perquè existeix una imatge igual o perquè no existeix al disc (tornar a dir que en el meu cas, sempre existirà al disc però que ho comprovo igualment per que ho diu a l'enunciat). La ruta que segueix l'excepció és la següent:

Dins addImage() de <BibliotecaImatges> creem l'obj a través de throw, si ocorre l'execpió aquesta viatja a través de throws 8signatura del mètode) fins al controlador que és d'es d'on es crida a aquesta funció. Arribats aquest punt, ens adonem que no volem gestionar el try/catch aqui i per tant tornem a signar el mètode amb el throws VisorException per enviar el tractament de l'excepció un mètode més enllà. Seguint una mica més arribem al mètode que ha cridat aquest mètode i, ara si, estem a VisorUB2 i hem de gestionar l'error aquí mitjançant

un bloc d'intent i captura perquè ens trobem en el lloc indicat per retornar els missatges d'error corresponents amb l'usuari.

Si anem un moment a la interface <InImageList> veiem que el diseny imposat ens obliga a gestionar l'excepció a addImage() pero no diu res de removeImage(). Això em dona certa llibertat per gestionar l'error de quan eliminem una imatge. M'explico:

- Per eliminar una imatge trobo molt més net i precís obligar a introduir un número dins un rank (una línia de codi) i utilitzar el contains() abans esmentat, que no pas anar arrossegant una excepció de mètode en mètode. De tota manera si ho hagués fet mitjançant excepcions la manera hagués estat la mateixa que he explicat en el cas d'afegir una imatge.

8. Expliqueu si heu fet servir la sobrecàrrega de mètodes a la classe CtrlVisor i detallar com

Al controlador he sobrecarregat addImage() i removeImage(), uns afegeixen i esborren imatges a la biblioteca i els altres fan el mateix en els àlbums. Tot i que els he sobrecarregat, la meva opinió és que, ja que uns mètodes interactuen amb els àlbums i els altres amb la biblioteca em sembla més entenedor posar noms diferents. En qualsevol cas és una opinió personal i els he sobrecarregat per satisfer les condicions de l'exercici. De la mateixa manera i seguint aquesta forma de treballar he sobrecarregat també addImage() i removeImage() a la classe <VisorUB2> ja que passava exactament el mateix.

Altres sobrecarregues en altres classes

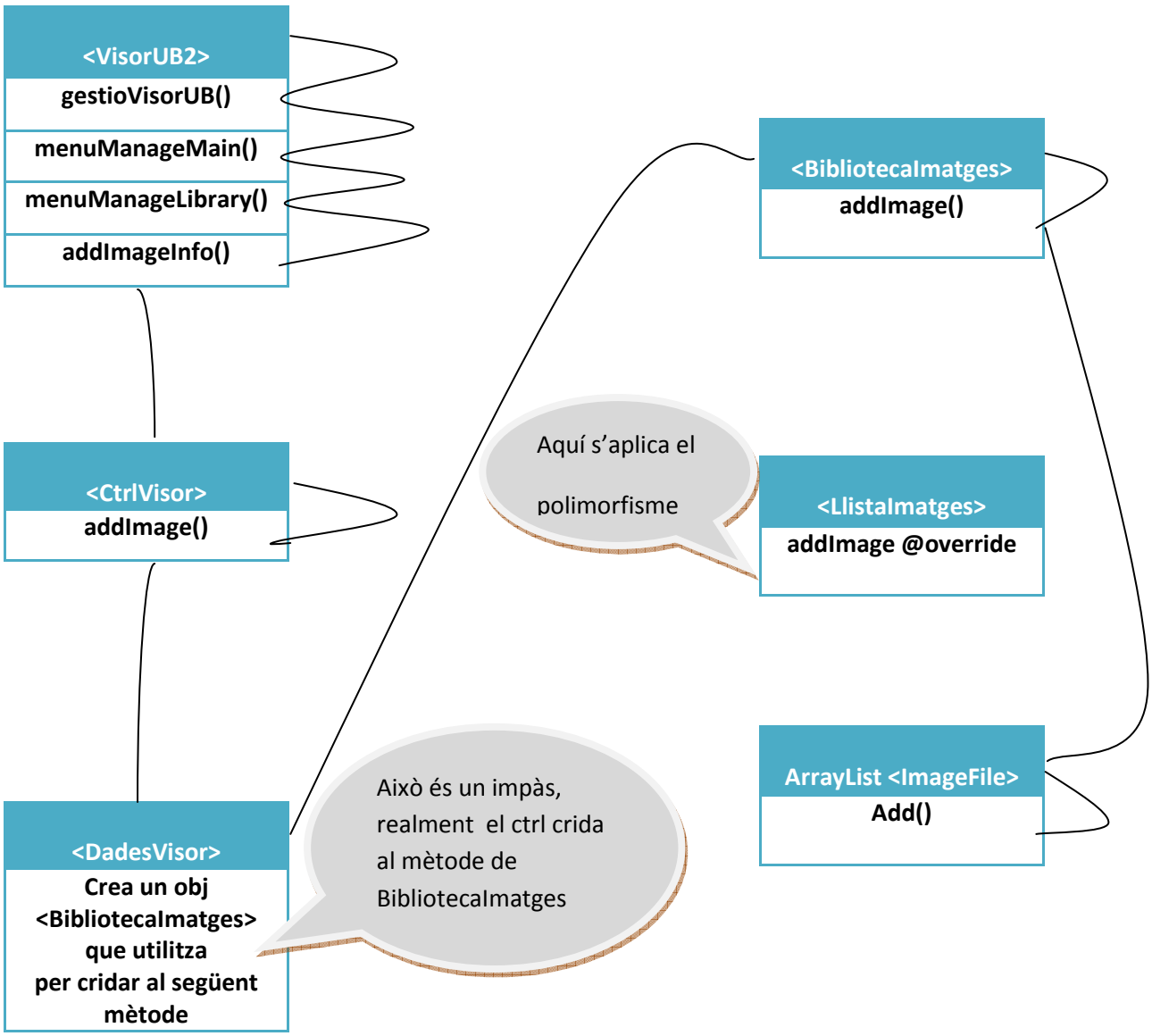
He sobrecaregat el constructor de <LlistaImatges>, un té un paràmetre que és la capacitat inicial (10) i aquest es crida des de <AlbumImatges>. L'altre sense paràmetre es crida des de BibliotecaImatges.

També he sobrecarregat el mètode removeImage() d'<AlbumImatges>, doncs aquest el teníem heredat de <LlistaImatges> i rebia un ImageFile. Ara la meva sobrecàrrega rep un enter que és la posició per eliminar per índex i no per objecte.

Gràcies al polimorfisme, ara, en temps d'execució el programa decideix a quin dels dos mètodes removeImage cridar, si al que té un enter com a paràmentre o al que té un <ImageFile>, jo per eliminar una imatge de l'àlbum faig servir el que rep un enter, el sobrecarregat.

La pregunta 9 està a la pròxima pàgina perquè el gràfic ocupa molt i necessito una plana sencera! Llavors les preguntes 10 i 11 es troben a continuació de la 9.

9. Completeu aquest dibuix per tal de mostrar el recorregut que fa el teu programa quan s'executa l'opció de afegir una imatge a la biblioteca. Especificar els mètodes que es criden en cadascuna de les classes. Fer servir fletxes i números per indicar l'ordre de les crides



Obs / a dades visor no cridem a cap mètode però el ctrl utilitza un obj seu per cridar a la biblioteca. S'observa que <BibliotecaImatges> crida directament a add() d'<ArrayList> fent ús de la sobreescriptura (polimorfisme)

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

Per provar el correcte funcionament la part de més proves ha estat la part de l'iterador que elimina totes les imatges dels àlbums que coincideixen amb la imatge que volem eliminar de la biblioteca.

He col·locat un iterador tal i com demana l'enunciat però he deixat comentada dins la funció la opció més fàcil de totes i que porta menys problemes que és el mètode removeAll() de les collections de java. Tot i que la he deixat comentada crec que és molt millor el removeAll, que ocupa menys i utilitza igualment un iterador dins el seu cos del mètode (implícitament).

També moltes proves han vingut de l'actualització de la portada d'un àlbum en què la seva imatge de portada ha estat eliminada des de la Biblioteca:

- Posem pel cas que tenim un àlbum "album1" que té com a imatge de portada la imatge de l'index [5] de la biblioteca que casualment es diu "imatge5". Si nosaltres eliminem la imatge5 de la biblioteca aquesta crida a la funció que té l'iterador que elimina la imatge de tots els àlbums. Però com que un dels àlbums "album1" tenia aquesta imatge com a portada ara hem de fer que la imatge de portada sigui la imatge que està situada a l'index [0] del mateix àlbum. Llavors si la imatge5 que eliminem era la única imatge de l'album, la portada de l'àlbum s'actualitza com a null. Aquesta acció també m'ha portat una miqueta de comprovacions extra.

Per seleccionar un àlbum per gestionar també vaig haver de crear un mètode que retornés un enter (índex de l'àlbum) per gestionar l'àlbum que toca segons la petició de l'usuari.

11. Observacions generals.

En les opcions d'editar la imatge de portada de l'àlbum o de canviar-li el títol he optat per no fer-ho junt. És a dir, ja que treballem amb menús he colocat un petit sub-menu que deixa triar 3 opcions:

1. Canviar portada
2. Editar títol
3. Sortir

(T'explico això perquè potser veus un sub-menu de més , però és perquè ja que estan fets m'estalvio crear un mètode del rollo menú i aprofito així el que estem utilitzant pels menús.)

Per guardar els fitxers de dades he capat les extensions a .dat i .obj com a úniques posibles perquè sinó pot ser un cachondeo.