Memòria

Introducció

L'objectiu d'aquesta pràctica és construir un visor d'imatges amb una interface gràfica (a llarg termini) i començant a crear-lo des de baix de tot de la programació de l'aplicació. En aquesta primera entrega volem poder crear una llista d'imatges que pugui ser guardada i recuperada en un fitxer extern mitjançant una vista per interactuar amb l'usuari i un model per gestionar l'aplicació.

Anàlisi

Com he esmentat a l'apartat anterior el primer que necessitem fer és tenir molt clar quin mòdul gestionarà l'aplicació i quin altre mòdul farà de mediador amb l'usuari. Dit això arribem a la conclusió que necessitem una vista per interactuar amb l'usuari i un model per gestionar l'aplicació.

Desenvolupament

Per desenvolupar la primera part de l'aplicació disposo d'unes classes i interfícies ja desenvolupades per un altre programador i tinc l'obligació de seguir un determinat disseny. Això implica una sèrie d'herències, interfaces i mètodes abstractes imposats "des de dalt" que he d'acoblar a l'aplicació per al seu correcte (únic) funcionament. Entre d'altres tenime mètodes per: afegir imatges, eliminar imatges, formatejar l'aplicació, comprovar l'existència d'una imatge, retornar una imatge, guardar les dades, recuperar les dades, etc. Tot està més explicat dintre del codi i a la part de preguntes d'aquest informe.

Resultats

El resultat de tot això ha estat una primera part de l'aplicació molt neta i ordenada on cada petita funcionalitat té un mètode que l'implementa seguint el concepte de modularitat explicat a teoria.

Secció de preguntes

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

Es demana construir un visor d'imatges que de moment es limita en guardar els objectes <Imatge> en un array que es podrà guardar en un fitxer .dat que serà la nostra "Base de dades". D'aquesta manera podrem recuperar les dades guardades d'una sessió anterior durant la sessió actual. A més cal construir-ne dues versions (una utilitzant array/tupla i l'altra fent servir <ArrayList>.

2. Explicar les classes implementades per mi.

Classe <Imatge> : representarà l'obj <Imatge> i se'n podran construir tantes com es vulgui. Aquests objectes seran guardats en un array (que s'explica en la classe següent). L'usuari també podrà anar eliminant aquests objectes si ho desitja. Aquesta classe conté els mètodes imprescindibles per interactuar directament amb l'obj <Imatge> (equals, toString, getters...). Hereda de la superClasse ImageFile que a la vegada hereta de File.

Classes <TaulaImatges> i <LlistaImatges>: Aquestes classes contenen l'array[] o ArrayList que enmagatzema les imatges. Contenen els mètodes necessaris i "algun" de suport per realitzar totes les operacions que calen fer sobre la llista d'imatges (afegir, eliminar, etc..) <LlistaImatges> hereta de la classe <ImageList> i <TaulaImatges> té una interfície <InImageList>.

Classe <VisorUB1>: Aquesta classe s'encarrega de crear i mostrar el menú a l'usuari. És la classe amb la que l'usuari final té interacció directa. En aquesta classe he creat els mètodes de suport que he necessitat per un bon desenvolupament del programa.

Classe <GestioVisorUB> : Aquesta és la classe que conté el main i permet triar a l'usuari quina versió del reproductorvol fer servir.

3. Explicar què s'ha pogut reutilitzar de la classe TaulaImatges per a la implementació de la classe LlistaImatges.

En el meu cas en particular primer he creat <TaulaImatges> i després amb les modificacions necessàries he creat <LlistaImatges>.

El canvi es que a LlistaImatges ens beneficiem de les funcions ja fetes de la classe ArrayList i només ens cal cridar-les.

Per citar un exemple, en el mètode clear(), a la classe TaulaImatges he de construir jo manualment la sentència iterativa que recorrerà l'array i anirà eliminant un per un els objectes, mentre que a <LlistaImatges> només em cal invocar al mètode clear() (es diu igual) de la classe <ArrayList> per aconseguir el mateix resultat de forma automàtica.

L'altra diferència crucial és la capacitat dinàmica de la classe ArrayList enfront l'array[] normal que és estàtic.

4. Explicar com has declarat l'atribut de la classe VisorUB i perquè

L'atribut de la classe VisorUB no pot ser privat ja que l'hem de cridar d'es d'una altra classe. Per tant una opció es declarar-lo 'public', o com en el meu cas, no posar-li res al davant, llavors es considera 'friendly' i es pot instanciar un obj des de qualsevol classe dins el mateix paquet. (En el nostre cas és ideal).

5. Explicar què cal fer per canviar el tipus d'implementació de la llista d'imatges en el VisorUB.

Per canviar d'una llista a una altra l'usuari trobarà un mini-menú de dues opcions només inicialitzar l'aplicació.

Segons quina sigui la seva entrada (1 o 2), aprofito la mateixa entrada de teclat per passar-la per paràmetre a VisorUB1 que és la classe on li donem vida al objecte. Es fa una comparació i si l'usuari ha entrat (1) anirà a parar al visor que conté l'array normal. Si prem (2) utilitzarà el visor amb <ArrayList>.

6. Dibuixar el diagrama de relacions entre les classes que has utilitzat a la teva pràctica. Incloure tant les classes implementades per tu com les que pertanyen a la llibreria UtilsProg2. No cal incloure la llista d'atributs i mètodes. Utilitzar la notació de la següent figura, per indicar que B és una classe que hereta de A i C conté una instància de la classe B, com al següent codi:

Aquest diagrama es troba al final del document a la última página!!

- Linia gruixuda indica herència
- Linia prima indica interfície
- Linia discontínua implica instanciació

7. Explicar quins són els atributs de la classe Imatge i perquè

imageName --> String que ens indica el nom de la imatge (no té perquè coincidir amb el nom del fitxer).

Només hem hagut de crear un atribut docs tots els altres els podem extreure directament de la superclasse i ens els donen mètodes ja implementats com getName(), etc...

8. Explicar com has implementat i on has utilitzat el mètode isFull() heretat de la classe ImageList

Al mètode isFull() el crida el mètode addImage() quan l'usuari vol afegir una imatge. isFull() comprova que si l'array ja ha arribat a 100 objectes i en cas que així sigui retornarà true i no es podrà afegir la imatge fins que n'eliminem, almenys, una.

Està implementat de la següent manera:

A <TaulaImatges> és un mètode boleà que retorna la comparació del nombre d'imatges actuals (getsize()) amb amb el tamany (length) de l'array que és 100.

A <LlistaImatges> també és un mètode boleà i retorna la comparació del getSize() amb MAX_IMAGES la constant que és 100. Aquí no podem comparar directament amb el tamany de l'array perquè aquest és dinàmic.

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

He anat creant les opcions del menú una a una sense passar a la següent fins que no ha estat absolutament robusta l'anterior.

Per aconseguir-ho he anat fent echo de tot el que feia per mostrar-ho per pantalla i quan així no n'he tingut prou aleshores he realitzat un debug per executar el codi pas a pas.

Una de les proves que he vist més importants ha estat que, en general, quan el programa espera un dígit en l'entrada de teclat per part de l'usuari i rep un caràcter no-numèric llavors peta. Per solucionar això al llarg de tot el meu programa, les entrades per teclat es reben com a String i llavors tenim funcions que comproven si l'entrada és un dígit, en cas contrari retorna un missatge d'error.

10. Segons la implementació de la classe <LlistaImatges>, si tenim dues imatges corresponents al mateix fitxer, quan cridem al mètode per eliminar un d'aquests fitxers eliminarà l'altre també o no?

En principi si ja que he creat el mètode equals() de la classe <Imatge> de forma que només compari el path i el nom de la imatge ja que si comparem tots els atributs el mètode retornaria sempre false perquè dues imatges no poden ser creades en el mateix instant de temps.

(Bé ara això no passaria perquè el lastModification() de moment retorna la data malament per que encara no treballem amb els arxius i sempre retorna la mateixa data de 1970.).

Però ara tenim el problema de que si un usuari elimina la seva imatge de la posició [2] i la imatge de la posició [6] és igual s'eliminarien totes dues imatges i no podem permetre que això passi ja que el desig de l'usuari al triar l'opció és que s'elimini la imatge de la posició que vol (i no més imatges).

Per solucionar això tinc un mètode imageExist() que entre d'altres funcionalitats és cridat per addImage() quan l'usuari vol afegir una imatge, i si la imatge ja existeix no la deixa afegir. En resum, per solucionar el problema impedeixo d'entrada que l'array tingui dos objectes iguals.

11. Observacions generals

He fet el programa el més robust possible perquè no peti en cas d'entrades per teclat errònies per part de l'usuari.

Es a dir, que si el programa espera un numero i li donem una lletra enlloc de petar entrega un missatge d'error.

Perquè això sigui possible, les entrades per teclat es reben com a string i no com a enter.

Això també ho he tingut en compte al Menú principal o no estava implementat així i per tant no era prou robust. Per a fer-ho he creat manualment un mètode getOption() personalitzat que retorna un obj tipus <OpcionsMenuPrincipal> després de comprovar que sigui un dígit i que es trobi dintre del rang adequat.

**** Manipulació del fitxer de dades *****

Per manipular el fitxer de dades deixant triar a l'usuari el path on el vol adreçar sería imprescindible utilitzar un entorn independent de la plataforma per exemple JFileChooser de SWING que és molt professional i fàcil d'implementar. Per aquest motiu escriure la ruta a mà crec que no és una solució bona i molt menys robusta. Jo de moment ho he solucionat de la següent manera:

He creat un directori fixe on es guardaran les dades tenint en compte tots els casos.

1. És independent del visor i del sistema operatiu: es poden crear i guardar dades des del Visor1 i recuperar-les des del Visor2. De la mateixa manera es poden crear dades desde linux i recuperar-les des de Windows o iOS (he solucionat el tema dels slashes utilitzant "File.separator" de la classe File).

2. Si es vol guardar dades noves i l'aplicació detecta que ja existeix un fitxer de dades, la primera vegada apareix un missatge preguntant si es volen sobreescriure les dades anteriors.

3. De forma anàloga si l'usuari vol recuperar dades antigues i el sistema detecta que hi ha dades actuals dades introduïdes a l'array, la primera vegada apareixerà un missatge dient que hi ha dades sense guardar i que si carrega les dades antigues es perdran les actuals.

Per fer aquestes tasques he creat un mètode de suport que explori el directori src/dades per saber si existeix el fitxer, i un boleà "savedRemember" evita que el missatge es repeteixi quan és innecessari, per exemple, si premem varies vegades seguides "recuperar" només ha de mostrar el missatge informatiu de sobreescriptura la primera vegada.

D'altra banda he creat la base de dades com a constant a <VisorUB1> així com el directori i el nom del fitxer ja que són atributs de classe que no canviaran al llarg del programa. (Sóc conscient del que vas dir que no volem variables globals per comoditat). Aqui està implementat pensant el millor disseny possible des del meu punt de vista.

Un altre detall és que el boleà savedRemember està implementat com a públic perquè en dues ocasions el crido des de <LlistaImatges>. En la propera entrega, quan puguem fer ús del controlador molts aspectes quedaran més organitzats.

En la propera pàgina es troba el diagrama de classes!

Diagrama de classes