PROJECTES DE PROGRAMACIÓ

Quadrimestre de tardor, curs 2019/2020

EL COMPRESSOR

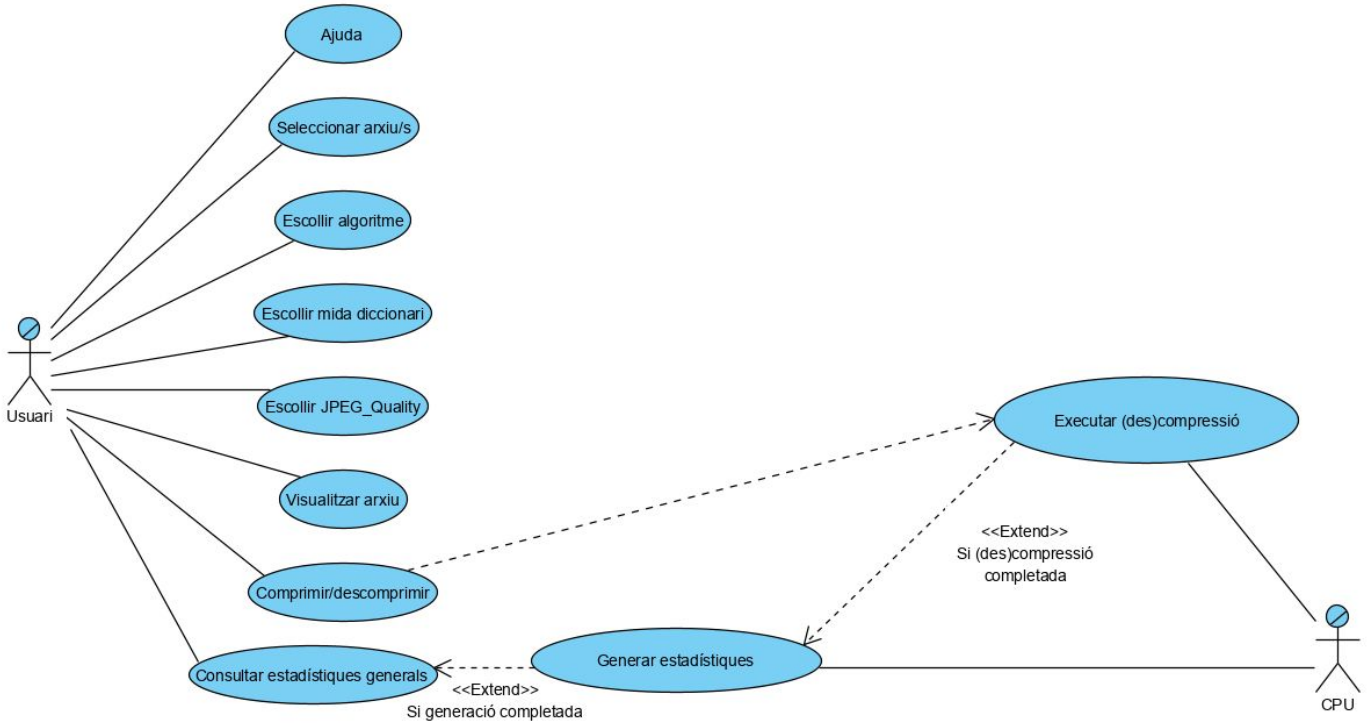
IBARS CUBEL, ALBERT	albert.ibars.cubel@est.fib.upc.edu
MUÑOZ BUSTO, ISAAC	isaac.munoz.busto@est.fib.upc.edu
CLEMENTE MARÍN, DANIEL	daniel.clemente.marin@est.fib.upc.edu
PÉREZ JOSENDE, ALEXANDRE	alexandre.perez.josende@est.fib.upc.edu

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Índex

1. CASOS D'ÚS

1.1. Esquema



1.2. Descripció

Nom cas d'ús:	**Seleccionar arxiu/s**
Descripció:	L'usuari selecciona un o més arxius que vol comprimir o descomprimir
Actors:	Usuari
Precondició:	El programa està obert però no s'està executant cap compressió ni descompressió
Diàleg típic:	1. L'usuari escull l'opció "Open" 2. El sistema presenta els directoris de l'usuari 3. L'usuari escull un directori 4. El sistema presenta el contingut del directori seleccionat 5. L'usuari escull un fitxer 6. El sistema obre un nou panel el qual conté: ● Algoritme(per escollir l'algoritme que utilitzarà la compressió, els algoritmes que surten depenen del tipus d'arxiu seleccionat) ● Parameter(per escollir la mida del diccionari en el cas dels algoritmes sense pèrdua o per escollir el JPEG_Quality pel cas del JPEG) ● Botó per visualitzar el fitxer ● Estadístiques sobre la compressió del fitxer 7. Apareix un nou botó al panel superior que depèn de si és un arxiu comprimit o no, el botó pot ser "Compress" o "Decompress".
Errors o vies alternatives:	● Si l'arxiu és un arxiu ja comprimit les opcions Algoritme i Parameter no es podran modificar. ● Si es tracta d'una carpeta no apareix el nou panel
Postcondició:	L'arxiu escollit està llest per comprimir o descomprimir

Nom cas d'ús:	**Escollir algoritme**
Descripció:	L'usuari selecciona amb quin algoritme vol comprimir l'arxiu
Actors:	Usuari
Precondició:	L'usuari ha seleccionat un arxiu compatible amb el nostre compressor i que no està comprès.
Diàleg típic:	1. El sistema li presenta les opcions que té depenent del tipus d'arxiu seleccionat mitjançant el desplegable "Algoritme" 2. L'usuari escull l'algoritme amb el desplegable "Algoritme"
Errors o vies alternatives:	Si l'arxiu ja està comprimit aquesta funcionalitat es troba bloquejada
Postcondició:	Algoritme seleccionat per poder executar la compressió amb ell

Nom cas d'ús:	**Escollir mida diccionari**
Descripció:	L'usuari selecciona quina és la mida del diccionari que utilitzarà l'algoritme de compressió escollit manualment o automàticament pel sistema
Actors:	Usuari
Precondició:	L'usuari ha seleccionat un arxiu i l'algoritme "LZ78" o "LZW"
Diàleg típic:	1. El sistema li presenta les opcions que té de diferents mides al desplegable "Parameter" depenent de l'algoritme seleccionat. 2. L'usuari escull la mida amb el desplegable "Parameter"
Errors o vies alternatives:	Si es tracta del LZSS aquesta funcionalitat es troba bloquejada ja que té una mida de diccionari fixa
Postcondició:	Mida del diccionari seleccionada per poder executar la compressió amb ella

Nom cas d'ús:	**Escollir JPEG_Quality**
Descripció:	L'usuari selecciona quina és la JPEG_Quality que utilitzarà l'algoritme de compressió JPEG
Actors:	Usuari
Precondició:	L'usuari ha seleccionat un arxiu i l'algoritme "JPEG""
Diàleg típic:	1. El sistema li presenta les opcions que té al desplegable "Parameter". 2. L'usuari escull la variable amb el desplegable "Parameter"
Errors o vies alternatives:	Si no s'escull cap s'utilitzarà DEFAULT
Postcondició:	JPEG_Quality seleccionat per poder executar la compressió amb JPEG

Nom cas d'ús:	**Visualitzar arxiu**
Descripció:	L'usuari té l'opció de visualitzar l'arxiu que vol comprimir o descomprimir
Actors:	Usuari
Precondició:	L'usuari ha seleccionat un arxiu
Diàleg típic:	1. L'usuari prem el botó Display 2. El sistema obre una nova finestra amb l'arxiu obert
Errors o vies alternatives:	● Si l'arxiu no és compatible amb el nostre programa no es visualitzarà correctament ● Si es tracta d'una imatge a comprimir apareix un altre botó per poder visualitzar la imatge que resultarà després de la compressió amb pèrdues
Postcondició:	Visualització de l'arxiu en una altra finestra

Nom cas d'ús:	**Comprimir/descomprimir**
Descripció:	L'usuari pot iniciar la compressió o la descompressió amb l'algoritme seleccionat
Actors:	Usuari
Precondició:	L'usuari ha seleccionat un arxiu compatible amb el nostre sistema
Diàleg típic:	1. L'usuari prem el botó "Compress" o "Decompress" 2. El sistema pregunta on guardar l'arxiu resultant i amb quin nom amb una finestra on podem navegar pels directoris del nostre ordinador 3. L'usuari selecciona directori on es guarda l'arxiu resultant, també pot crear una carpeta indicant el nom que li vol donar i l'arxiu resultant es quedarà a dins 4. S'executa la (des)compressió
Errors o vies alternatives:	El botó serà "Compress" o "Decompress" depenent di l'arxiu ja està comprimit o no
Postcondició:	El programa passa al cas d'ús "Executar (des)compressió"

Nom cas d'ús:	**Consultar estadístiques generals**
Descripció:	L'usuari pot visualitzar les estadístiques de totes les compressions fetes pel compressor
Actors:	Usuari
Precondició:	L'usuari ha comprimit algun arxiu
Diàleg típic:	1. L'usuari prem el botó "Statistics" 2. El sistema obre una nova finestra amb les següents estadístiques de totes les compressions executades: ● Ratio de compressió ● Espai estalviat ● Bytes llegits ● Bytes escrits ● Temps trigat en les compressions/descompressions ● Compressió per segons
Errors o vies alternatives:	El botó no es trobarà visible fins que no s'executi cap compressió
Postcondició:	Visualització de les estadístiques totals del programa en una altra finestra

Nom cas d'ús:	**Ajuda**
Descripció:	L'usuari disposa d'un document d'ajuda per resoldre els dubtes del funcionament del programa
Actors:	Usuari
Precondició:	Programa obert
Diàleg típic:	1. L'usuari prem el botó "Help" 2. El sistema obre una nova finestra amb un document del manual d'usuari del programa
Errors o vies alternatives:	
Postcondició:	Visualització del manual d'usuari del programa en una altra finestra

Nom cas d'ús:	**Executar (des)compressió**
Descripció:	S'executa la compressió o la descompressió amb l'algoritme seleccionat
Actors:	CPU
Precondició:	L'usuari ha pres el botó de "Compress" o "Decompress"
Diàleg típic:	1. El sistema executa l'algoritme escollit per l'usuari o automàticament per l'arxiu seleccionat
Errors o vies alternatives:	El botó serà "Compress" o "Decompress" depenent di l'arxiu ja està comprimit o no
Postcondició:	Arxiu comprimit o descomprimit es guarda a l'ordinador

Nom cas d'ús:	**Generar estadístiques**
Descripció:	Es generen les estadístiques de compressió per l'arxiu seleccionat i les totals del programa s'actualitzen
Actors:	CPU
Precondició:	L'usuari ha comprimit(descomprimit un arxiu
Diàleg típic:	1. El sistema genera les estadístiques i les guarda 2. Les estadístiques de l'arxiu comprimit es podran visualitzar quan es seleccioni aquell arxiu i les generals amb un botó al panel superior.
Errors o vies alternatives:	Si hi ha algun error en la compressió relacionat amb les estadístiques el sistema llença una excepció i no guarda les estadístiques
Postcondició:	Les estadístiques queden guardades al sistema

2. DISSENY

Degut a que el tamany dels diagrames és força gran i la seva apreciació detallada es fa difícil, recomanem la seva visualització obrint la imatge de cada diagrama que es troba en la carpeta DOCS del projecte.

2.1. Diagrama de persistència

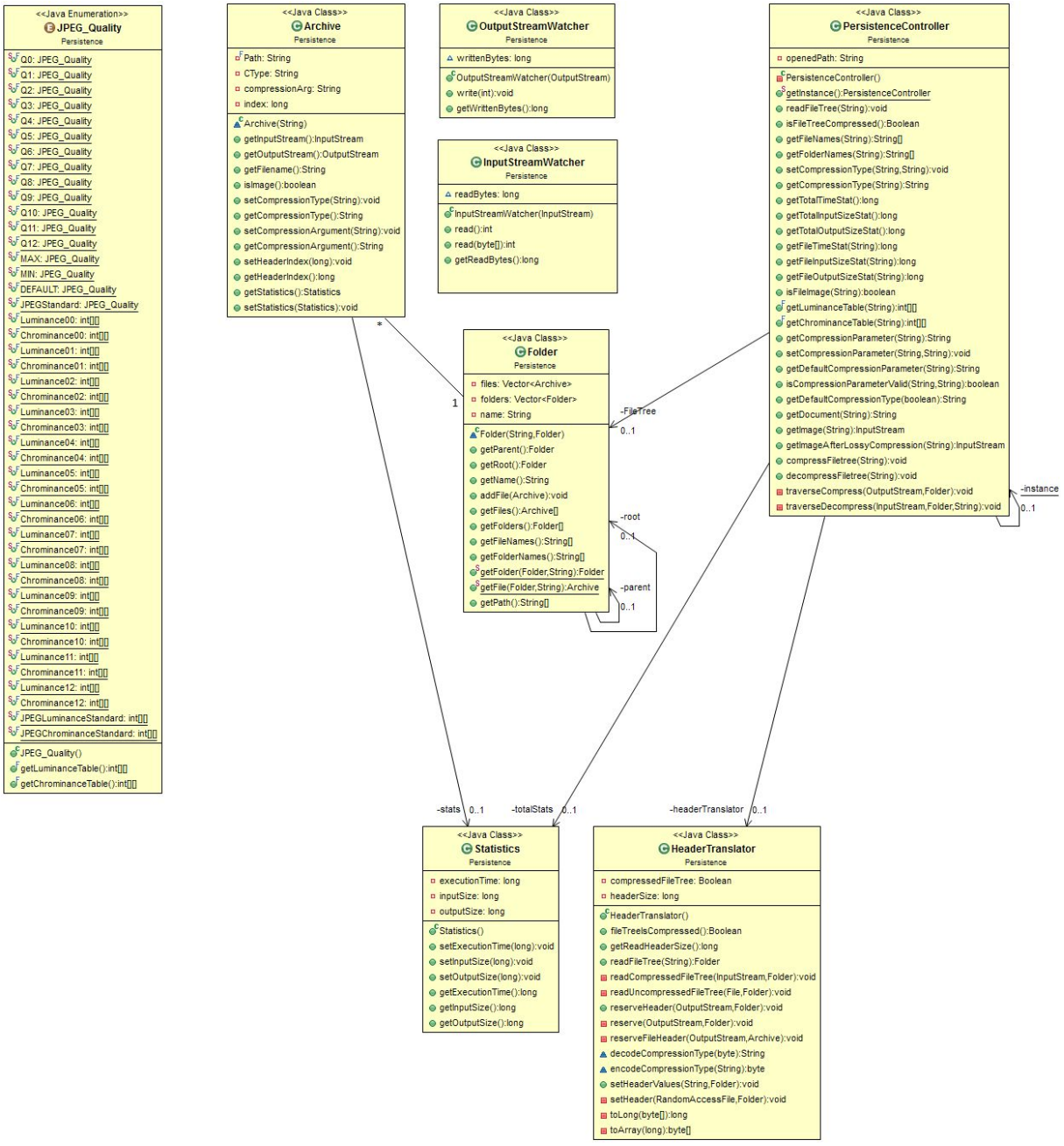


Diagrama de persistència

2.2. Diagrama de domini

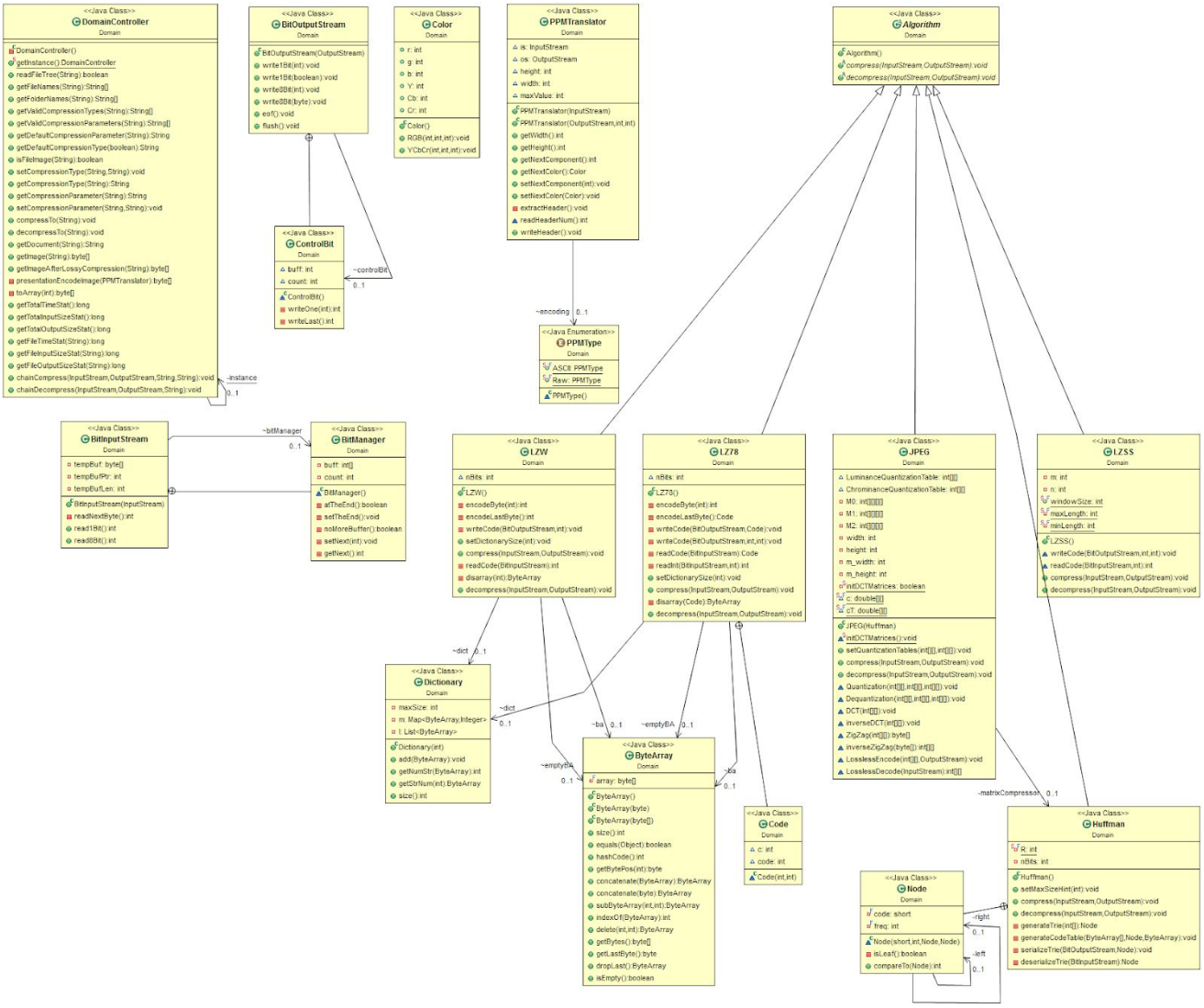


Diagrama de domini

2.3. Diagrama de presentació

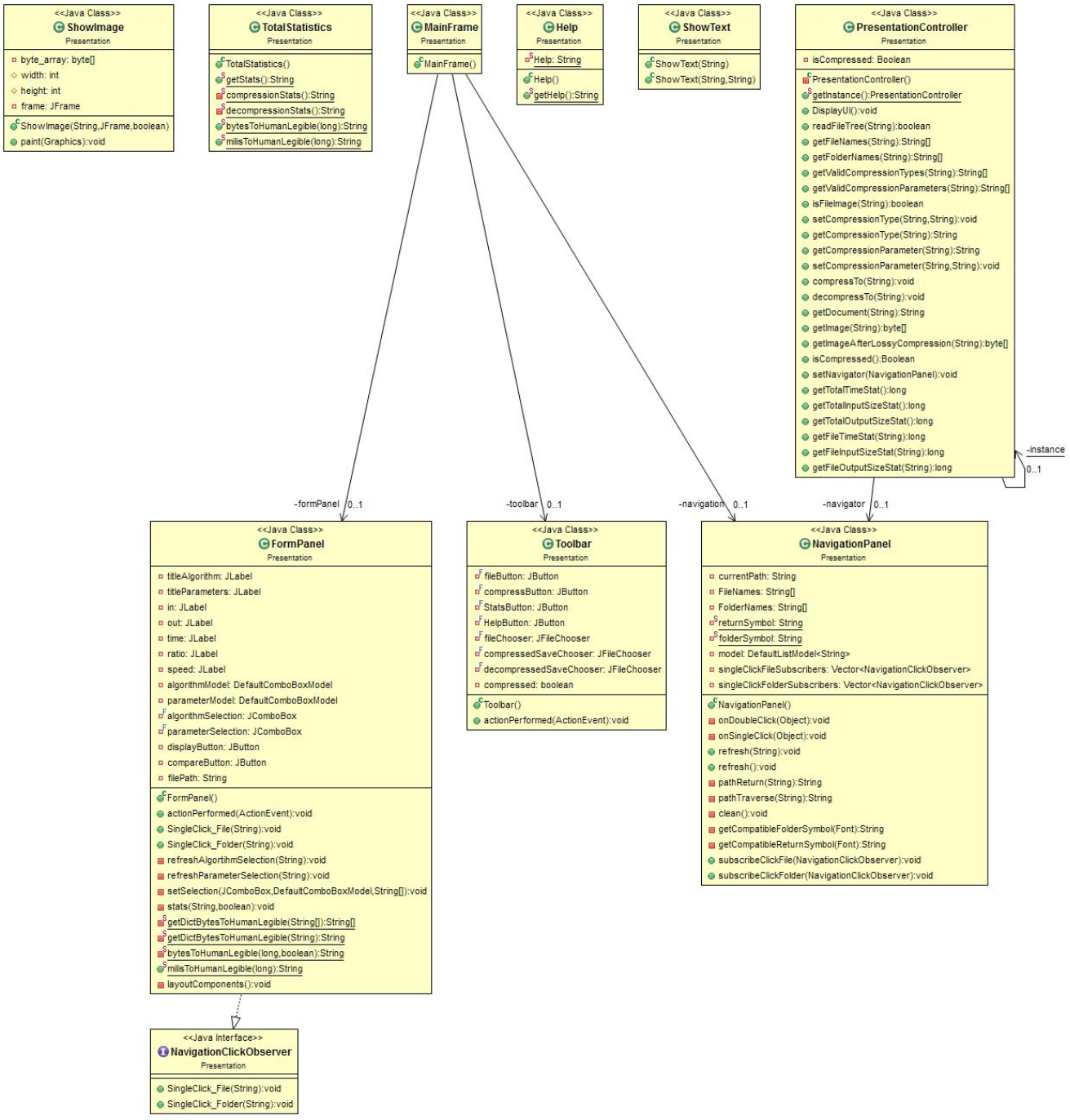


Diagrama de presentació

2.4. Explicació de les classes

Algorithm: Interfície que defineix els atributs i els mètodes que tenen els algoritmes implementats en comú.

LZ78: Implementa l'algoritme LZ78, que consisteix en agafar el fitxer d'entrada i amb un diccionari anar guardant les cadenes de caràcters que es van formant i es van repetint tenint en compte les ja formades mitjançant un diccionari.

LZW: Implementa l'algoritme LZW el qual es basa en l'anterior LZ78, que consisteix en la creació d'un diccionari el qual al principi s'inicialitza amb tots els valors ASCII i es va emplenant de cadenes de caràcters a mesura que es llegeix l'arxiu d'entrada i el qual no s'inclou al fitxer comprimit ja que al descomprimir es pot reconstruir fàcilment.

LZSS: Implementa l'algoritme LZSS que consisteix en anar llegint l'entrada mitjançant una finestra de mida fixa que va buscant coincidències i codifica tuples amb la posició de la coincidència trobada i la mida de la cadena. Es pot codificar com a literal si no troba coincidència ja que ocupa menys com a literal que com a tupla.

JPEG: Implementa l'algoritme JPEG que es basa en la simplificació del color d'uns quants píxels i la posterior compressió amb Huffman del resultat. Al descomprimir l'arxiu es perd qualitat casi imperceptible per l'ull humà.

Huffman: Algoritme cridat per l'algoritme JPEG que s'encarrega de comprimir utilitzant les freqüències de cada símbol mitjançant un arbre, els caràcters més freqüents tenen una codificació més curta.

Dictionary: Diccionari propi de ByteArrays implementat amb un hashmap i una llista.

ByteArray: Estructura formada per un array de bytes.

BitInputStream: Classe encarregada de llegir l'input bit a bit i passar les dades que llegeix a l'algoritme que s'executarà.

BitManager: Classe privada del BitInputStream que controla els fluxos de bits de entrada.

BitOutputStream: Classe encarregada de llegir els resultats dels algoritmes de bit a bit i passar les dades que llegeix a un nou arxiu.

ControlBit: Classe privada del BitOuputStream que controla els fluxos de bits de sortida.

PPMTranslator: Classe que serveix de parser de ppm binaris i ASCII i que escriu en format ppm binari. Permet llegir tan colors com components individuals d'aquests colors i també escriure els colors o les seves components individuals.

Color: Classe que guarda i transforma colors RGB/YCbCr.

JPEG_Quality: Classe encarregada d'enumerar les diferents qualitats disponibles del jpeg i proporcionar els mapejos d'aquestes qualitats a les corresponents matrius de quantificació per la luminància i la crominància.

DomainController: Controlador de domini, singleton. Implementa diferents funcions que comuniquen la capa de domini amb Persistencia i Presentació.

Archive: Classe que representa un arxiu en la jerarquía de fitxers i que conté totes les propietats i mètodes propis de l'arxiu.

Folder: Classe que representa un carpeta en la jerarquía de fitxers i que conté totes les propietats i mètodes propis de la carpeta.

HeaderTranslator: Classe que implementa la codificació de la capçalera d'arxius comprimits. Codifica la jerarquia d'arxius i carpetes durant la compressió i es capaç de decodificar-la posteriorment permetent l'accés aleatori a qualsevol arxiu compress.

InputStreamWatcher: Classe decoradora de InputStream que implementa una consultora que retorna el número de bytes llegits.

OutputStreamWatcher: Classe decoradora de OutputStream que implementa una consultora que retorna el número de bytes escrits.

PersistanceController: Controlador de persistencia, singleton. Implementa diferents funcions que comuniquen la capa de Persistencia amb Domini.

Statistics: Clase que emmagatzema les diferents dades de espai (en bytes) i temps (en ms) usades per a la generació d'estadístiques. Conté getters i setters

FormPanel: Classe que conté tots els elements (i la implementació de les seves funcionalitats) que apareixen en la finestra de propietats de la interfície.

Help: Classe encarregada de mostrar el manual d'ajuda en cas de que l'usuari ho sol·liciti

MainFrame: Classe que representa la finestra que engloba el conjunt de la interfície i que serà extesa per les classes que formen cada part de la interfície (FormPanel, Toolbar i NavigationPanel).

NavigationClickObserver: Interfície que han d'implementar tots els observadors de NavigationPanel que vulguin rebre notificacions cada vegada que l'usuari prem un arxiu o carpeta al navegador.

NavigationPanel: Classe que conté tots els elements (i la implementació de les seves funcionalitats) que apareixen en la finestra de navegació de la interfície.

PresentationController: Controlador de presentació, singleton. Implementa diferents funcions que comuniquen la capa de Domini amb Presentacio.

ShowImage: Classe encarregada de mostrar una imatge determinada que es vol visualitzar en la pròpia interfície

ShowText: Classe encarregada de mostrar una document de text determinat que es vol visualitzar en la pròpia interfície

Toolbar: Classe que conté tots els elements (i la implementació de les seves funcionalitats) que apareixen en la barra d'eines de la interfície.

TotalStatistics: Classe encarregada de mostrar les estadístiques totals de l'última compressió/descompressió executada en cas de que l'usuari ho demani.

2.5. Patrons de disseny

En la implementació del programa hem seguit els següents patrons de disseny:

Singleton: S'utilitza per al controlador de cada capa ja que necessitem que només tinguin una instancia.

Decorador: S'utilitza en InputStreamWatcher que decora FilterInputStream de manera que la funció de llegida també compta els bytes llegits i també en OutputStreamWatcher que decora FilterOutputStream de manera que la funció de escriptura també conta els bytes escrits.

Observador: L'utilitzem a la capa Presentació amb l'objectiu de que les accions que es fan a NavigationPanel es notifiquin a FormPanel. Els observadors han d'implementar l'interfície NavigationClickObserver.

3. ESTRUCTURES DE DADES

3.1. Capa de persistència

Vectors a Folder: Per a representar el contingut d'una carpeta i així poder representar la jerarquía d'arxius del sistema. Hem creat dos tipus de vectors: un on emmagatzarem tots els arxius (no carpetes) de la carpeta i un altre on emmagatzarem totes les carpetes que es troben dins de la carpeta instanciada.

Donada aquesta estructuració del contingut de las carpetes, el sistema de fitxers quedarà estructurat com un **arbre** d'arxius i carpetes.

En aquesta capa no hem utilitzat cap estructura de dades explícita més, però volem comentar el cas del HeaderTranslator ja que funciona com una estructura de dades implícitament.

HeaderTranslator escriu la capçalera de l'arbre d'arxius i un dels camps que tenen els arxius (no carpetes) es un long (8 bytes) que conté el nombre de bytes des de l'inici del arxiu comprimit fins el primer byte de les dades comprimides d'aquell arxiu.

Aleshores quan es demana accés aleatori a un arxiu que esta comprimit no fa falta descomprimir tot el comprimit sino que es desplaça l'InputStream el nombre de bytes que indiqui el long que ve donat a la capçalera i es descomprimeix desde aquell byte. L'algoritme és capaç de saber quan ha de parar de descomprimir per que la classe BitInputStream llegeix bit a bit i detecta el final d'aquell arxiu.

D'aquesta manera actuem com si el stream fos una estructura de dades (més concretament com si el disc fos un array i nosaltres sabem l'índex d'inici de cada arxiu).

3.2. Capa de domini

Quan esmentem que hem utilitzat un Dictionary per buscar coincidències farem referència a l'estructura formada per un HashMap (que té un ByteArray com a clau i un cert codi com a valor) i una List de ByteArrays (que utilitzarem per buscar una ByteArray associat a un codi determinat).

Dictionary al LZ78 i LZW: Hem elegit aquesta estructura per poder buscar coincidències quan comprimim i obtenir el ByteArray codificat quan descomprimim en temps constant O(1). El seu tamany el definirem arbitràriament i influirà en el grau de compressió.

ByteArray al LZ78 i LZW: Hem elegit aquesta estructura ja que treballem sobre l'input i l'output a nivell de byte i un vector on cada posició és un byte ens facilita molt la feina quan realitzem operacions (p.e. concatenació o consulta) sobre aquests bytes. El seu tamany serà variable depenent del nombre de bytes que tingui la coincidència.

ByteArray al LZSS: Encara que sigui l'única estructura de dades utilitzada en aquest algoritme farem servir ByteArray per a dues funcionalitats diferents.

En primer lloc, ens servirà per a representar la **finestra corredissa** on buscarem les coincidències. El fet de implementar-la com a ByteArray ens permetrà fer accessos aleatoris amb cost O(1) i realitzar operacions de inserció i eliminació de conjunts de bytes de manera més òptima. El seu tamany l'assignarem arbitràriament i serà un factor determinant pel que fa al grau de compressió de l'algoritme.

En segon lloc, ens servirà per a representar el **buffer de coincidències** on anirem guardant la coincidència actual. Obtindrem un gran avantatge en implementar-lo com a ByteArray ja que com treballem sobre l'input i l'output a nivell de byte ens facilitarà molt operacions com la inserció de un

nou byte al buffer o com una consulta aleatòria. El seu tamany serà dinàmic ja que anirà variant segons els bytes que formin part de la coincidència actual.

Matriu al JPEG: Utilitzem matrius per representar la imatge. La imatge la dividirem en submatrius de 8x8. Això comporta que, quan haguem de seleccionar una posició de la imatge, determinem 4 valors on el dos primers [][] serviran per seleccionar la submatriu 8x8 i els dos últims [][] faran referència a la posició dins la submatriu. Utilitzarem 3 matrius d'aquest tipus (matriu[][][][]) on cadascuna farà referència a una component del color. El fet d'utilitzar matrius farà possible l'accés aleatori a un cost constant. El tamany de les matrius serà estàtic i vindrà directament determinat pel tamany de la imatge.

Array de bytes al JPEG: Utilitzem arrays de bytes per guardar dades generades per diferents funcionalitats que apareixen durant l'execució de l'algoritme. Hem utilitzat array de bytes en comptes de el tipus ByteArray creat per nosaltres perquè en aquest algorisme no usem cap mètode dels implementats en la classe ByteArray i, per tant, el usar aquesta classe ens generaria dependències innecessàries. D'aquesta manera (en estructurar les dades en un array de bytes) podrem accedir-hi aleatòriament en un cost constant.

ByteArrayInputStream/ByteArrayOutputStream al Huffman: Utilitzem aquesta estructura per traduir el array a stream per poder utilitzar-lo amb les funcions de l'algoritme Huffman (ja que hem de passar un stream per aconseguir que el Huffman sigui agnòstic respecte les dades que li passem).

Trie al Huffman: Hem utilitzat un trie per estructurar la codificació de les coincidències. El trie estarà format per nodes de tipus Node que contindran informació sobre el caràcter ASCII al que fa referència, la freqüència amb la que apareix aquest codi i quins son els nodes fills. Hem elegit aquesta estructura ja que ens permet realitzar consultes en cost O(m) on "m" és la longitud del codi. El tamany del trie serà dinàmic i anirà augmentant a mesura que anem tractant la imatge.

PriorityQueue al Huffman: Hem elegit una PriorityQueue per poder ordenar els arbres segons la freqüència d'aparició en el input. Encara que el cost d'ordenació serà el mateix que si utilitzéssim un vector, hem elegit una PriorityQueue ja que ens sembla la opció més intuïtiva. El seu tamany serà fixe ja que un cop inicialitzada contindrà els nodes arrel de 256 tries (un per cada símbol ASCII).

ByteArray/Array de bytes al Huffman: Com hem comentat en els apartats anteriors, utilitzarem aquestes dues estructures per emmagatzemar dades de manera seqüencial i de manera que la seva consulta es pugui dur a terme en cost O(1). Utilitzarem ByteArray quan vulguem utilitzar un dels seus mètodes. Altrament, utilitzarem un array de bytes.

3.3. Capa de presentació

Arrays de strings a NavigationPanel: On emmagatzarem els noms de les carpetes i arxius que es trobin en el nivell del la jerarquía de arxius en el que estem navegant. És un contenidor que tindrà generalment pocs elements i un array és més que suficient per contenir el llistat de noms amb ordre alfabètic.

Vector de NavigationClickObservers a NavigationPanel: Usat per al patró observer, quan un usuari clica un arxiu o una carpeta al navegador del compressor, els elements d'aquest vector seran avisats. Podríem haver usat una llista o qualsevol tipus similar, que permeti l'accés seqüencial i afegir nous elements fàcilment.

DefaultListModel de strings a NavigationPanel: Usat pel JList, genera el JList de Strings (noms de carpeta i arxius que es veu a la interfície).

Array de bytes a ShowImage: Emmagatzema l'amplada (primers 4 bytes) i la alçada (seguents 4 bytes) i els píxels en RGB (un byte per component RGB) de la imatge que es mostrarà per la finestra emergent de visualització.

4. DESCRIPCIÓ DELS ALGORITMES

El nostre projecte comprimeix i descomprimeix arxius mitjançant diferents algoritmes:

Algoritme LZ78 *(Lempel-Ziv, 1978)*
Algoritme LZSS *(Lempel-Ziv-Storer-Szymansk, 1982)*
Algoritme LZW *(Lempel-Ziv-Welch, 1984)*
Algoritme JPEG *(Joint Photgraphic Experts Group, 1992)*
Algoritme Huffman *(David A. Huffman, 1952)*

4.1. LZ78

L'algorisme LZ78 aconsegueixen compressió substituint les ocurrències repetides de dades per referències a un diccionari que es crea basant en flux de dades d'entrada. Cada entrada del diccionari es de la forma [...] = {index, character}, on index és l'índex d'una entrada de diccionari anterior, i el caràcter s'afegeix a la cadena representada pel diccionari[index]. Per a cada caràcter del flux d'entrada, es busca el diccionari una coincidència: {últim índex coincident, caràcter}.

Si es troba una coincidència, el darrer índex de concordança es defineix en l'índex de l'entrada coincident, i no es produeix res. Si no es troba una coincidència, es crea una nova entrada de diccionari: diccionari [següent índex disponible] = {últim índex de concordança, caràcter} i l'algorisme produeix l'índex de coincidència últim, seguit de caràcter, després es restableix l'últim índex de coincidència = 0 i augmenta el següent índex disponible.

Un cop el diccionari estigui complet, no s'afegeixin més entrades. Quan s'arriba al final del flux d'entrada, l'algoritme produeix l'últim índex coincident.

4.2. LZSS

El LZSS es un algoritme que millora el LZ77 mitjançant un indicador d'un bit per indicar si el següent fragment de dades és un literal o una parella de distància-longitud i l'ús de literals per si una parella de longitud-distància seria més llarga.

On els algorismes LZ77 aconsegueixen compressió substituint les ocurrències repetides de dades per referències a una sola còpia d'aquestes dades existents anteriorment al flux de dades no comprimides.

4.3. LZW

LZW és un algorisme basat en LZ78 que utilitza un diccionari preinicialitzat amb tots els possibles caràcters (símbols) o emulació d'un diccionari preinicialitzat.

La millora principal de LZW és que quan no es troba una coincidència, s'assumeix que el caràcter actual del flux d'entrada és el primer caràcter d'una cadena existent al diccionari (ja que el diccionari s'inicialitza amb tots els caràcters possibles), de manera que només l'última coincidència es produeix un índex (que pot ser l'índex de diccionari preinicialitzat corresponent al caràcter d'entrada anterior (o inicial)).

4.4. JPEG

JPEG utilitza una forma de compressió amb pèrdues basada en la transformada de cosinus discreta (DCT). Aquesta operació matemàtica converteix cada fotograma / camp de la font de vídeo del domini espacial (2D) en el domini de freqüència (per exemple un domini de transformació). Un model perceptiu basat en el sistema psicovisual humà descarta informació d'alta freqüència, és a dir, transicions intenses en intensitat i color. Al domini de transformació, el procés de reducció de la informació s'anomena quantització.

El mètode de compressió sol perdre's, és a dir, que es perd alguna informació original de la imatge i no es pot restaurar, afectant possiblement la qualitat de la imatge.

4.5. Huffman

La tècnica funciona creant un arbre binari de nodes. Un node pot ser un node de fulla o un node intern. Inicialment, tots els nodes són nodes de fulla, que contenen el símbol en si, el pes (freqüència d'aparició) del símbol i, opcionalment, un enllaç a un node parent que facilita la lectura del codi (a la inversa) a partir d'un node de fulla.

El procés s'inicia amb els nodes de full que contenen les probabilitats del símbol que representen. Aleshores, el procés agafa els dos nodes amb menor probabilitat i crea un nou node intern que té aquests dos nodes com a fills. El pes del nou node s'estableix en la suma del pes dels fills. A continuació, apliquem el procés de nou, al nou node intern i als nodes restants (és a dir, excloem els dos nodes de full), repetim aquest procés fins que només queda un node, que és l'arrel de l'arbre de Huffman.

5. CLASSES IMPLEMENTADES PER CADA MEMBRE

5.1. Daniel

FONTS/Domain/Dictionary.java

FONTS/Domain/LZ78.java

FONTS/Presentation/MainFrame.java

FONTS/Presentation/Toolbar.java

5.2. Albert

FONTS/Domain/BitInputStream.java

FONTS/Domain/BitOutputStream.java

FONTS/Domain/LZW.java

FONTS/Presentation/FormPanel.java

FONTS/Presentation/ShowImage.java

FONTS/Presentation/ShowText.java

FONTS/Testing/UnitTest_JPEG/HuffmanStub.java

FONTS/Testing/UnitTest_JPEG/UnitTest.java

FONTS/Testing/UnitTest_JPEG/UtilsTest.java

5.3. Isaac

FONTS/Domain/ByteArray.java

FONTS/Domain/LZSS.java

FONTS/Presentation/Help.java

5.4. Alexandre

FONTS/Domain/JPEG.java

FONTS/Domain/Huffman.java

FONTS/Domain/Algorithm.java

FONTS/Domain/Color.java

FONTS/Domain/DomainController.java

FONTS/Domain/PPMTranslator.java

FONTS/Persistence/Archive.java

FONTS/Persistence/Folder.java

FONTS/Persistence/HeaderTranslator.java

FONTS/Persistence/InputStreamWatcher.java

FONTS/Persistence/JPEG_Quality.java

FONTS/Persistence/OutputStreamWatcher.java

FONTS/Persistence/PersistenceController.java

FONTS/Persistence/Statistics.java

FONTS/Presentation/NavigationClickObserver.java

FONTS/Presentation/NavigationPanel.java

FONTS/Presentation/PresentationController.java

FONTS/Presentation/TotalStatistics.java

FONTS/MainDriver.java

FONTS/Testing/ExtraTests/HuffmanTest.java

FONTS/Testing/ExtraTests/IOTest.java

FONTS/Testing/ExtraTests/LZ78Test.java

FONTS/Testing/ExtraTests/LZSSTest.java

FONTS/Testing/ExtraTests/LZWTest.java

FONTS/Testing/UnitTest_JPEG/InputStreamStub.java

FONTS/Testing/UnitTest_JPEG/OutputStreamStub.java

FONTS/Testing/CustomTestListener.java

FONTS/Testing/ExtraTestsDriver.java

FONTS/Testing/UnitTestJPEGDriver.java