Segona Entrega
Documentació de classes, algorismes, estructures de dades i jocs de prova.

Ignasi Sant Albors (ignasi.sant)

Joaquim Ferrer Sagarra (joaquim.ferrer)

Jordi Garcia Aguilar (jordi.garcia.aguilar)

ÍNDEX

Introducció

Aquest document s'ha fet per ajuntar les explicacions de les classes, les explicacions dels algoritmes i la justificació de les estructures de dades usades per a facilitar la comprensió. **És a dir, de la rúbrica d'avaluació conté els apartats documentació del codi, estructures de dades i documentació dels jocs de prova.** Conté diagrames per a facilitar la comprensió de l'estructura.
Està escrit a mode de documentació general del projecte per explicar i justificar les decisions preses en els diferents estadis del desenvolupament. El redactem també perquè, donades les hores de treball i càrrega de feina que ens ha portat tota aquesta pràctica, creiem necessari explicar totes les decisions que hem pres i documentar com s'han implementat perquè la comprensió del codi sigui més senzilla i es pugui valorar i tenir en compte perquè les coses estan implementades com ho estan, podent donar més èmfasi a allò que realment ha sigut important i cabdal en el desenvolupament d'aquest projecte.

Capa de Presentació.

Aquesta capa conte les classes de presentació. Són controlador de presentació i les vistes de la interfície Gràfica.

Main

Classe Driver

Aquesta és la classe que conté el mètode main i per tant és el començament del programa. Des del mètode main s'inicialitza una instància del controlador de presentació **IOUtils** i es crida al mètode run().

Classe Driver
Autor: Ignasi Sant Albors
Mètodes públics:

- **main():** Crida al mètode run() de IOUtils

Controlador de presentació

IOUtils

Classe IOUtils
Autor: Ignasi Sant Albors
Mètodes públics:

- **IOUtils():** Constructora de la classe
    - **Pre:** -
    - **Post:** S'inicialitzen les variables de la vista principal i del controlador de Domini(Fitxer).
- **run():**
    - **Pre:** S'ha cridat al mètode run des del main.
    - **Post:** Es fa visible la vista principal que obra la finestra de la interficie gràfica i usant els inputs del usuari inicia l'acció pertinent cridant al controlador de Domini
- **SetType(int type):** Mètode per definir si s'ha de tractar un fitxer o carpeta.
- **setInputFile(String file):** Mètode per definir el path de l'arxiu a tractar.
- **setOutputFile(String file):** Mètode per definir el path on es guardara l'arxiu tractat.
- **setAction(int action):** Mètode per definir si es comprimeix o descomprimeix el fitxer.
- **setAlgorithm(int algo):** Mètode per definir l'algoritme de compressió/descompressió
- **getStats():** Mètode que retorna un String array amb les estadístiques del fitxer tractat.
- **getAllStats():** Mètode que retorna un String array amb les estadístiques històriques.
- **getCompare():** Mètode que retorna un String array amb el fitxer original i el tractat després de comprimir i descomprimir.
- **getType():** Mètode que retorna si el fitxer a tractat és un arxiu o directori.

Explicació i implementació de la classe:
La classe IOUtils actua com a controlador de la capa de presentació i s'encarrega de la comunicació amb l'usuari a través de les vistes que implementen la interfície gràfica. També s'encarrega de la comunicació amb la capa de domini, concretament comunicant-se amb el seu controlador, la classe Fitxer. Per tal de mantenir l'arquitectura de 3 capes només es passen tipus de variables simples (int, String, etc)entre els controladors.

Vistes

Son les responsables de la gestió de la interfície gràfica. Interpreten les accions de l'usuari i les comuniquen a la capa de presentació. Capturen i solucionen alguns errors per aconseguir una correcta execució del programa.

VistaPrincipal

Classe VistaPrincipal
Autor: Ignasi Sant Albors
Mètodes públics:

- **VistaPrincipal(IOUtils pIOUtils):** Constructora classe VistaPrincipal. Inicialitza la variable de tipus IOUtils.
- **hacerVisible():** Mètode que fa visible el frame que conté la Interficie Gràfica del programa.
- **update():** Mètode que repinta el frame que conté la Interfície Gràfica del programa.

- **volverHome():** Mètode que repinta el frame que conté conté la Interfície Gràfica del programa amb els continguts inicials d'aquest.

VistaSelAlg

Classe VistaSelAlg
Autor: Ignasi Sant Albors
Mètodes públics:

- **VistaSelAlg(IOUtils plOUtils, VistaPrincipal vp, Boolean jpeg):** Constructora de la classe VistaSelAlg. Inicialitza els atributs de tipus IOUtils i VistaPrincipal. La variable boolean servirà per capturar possibles excepcions.
- **getPanelComprimir():** Mètode que retorna un JPanel on hi ha tots els components de la interfície gràfica referents a la selecció d'un algoritme per a la compressió.

VistaInfo

Classe VistaInfo
Autor: Ignasi Sant Albors
Mètodes públics:

- **VistaInfo(IOUtils plOUtils, Boolean c, VistaPrincipal vp):** Constructora de la classe VistaInfo. Inicialitza els atributs de tipus IOUtils i VistaPrincipal. El Boolean passat com a paràmetre indica si es visualitzarà la informació d'una compressió o descompressió
- **getPanelInformacio():** Mètode que retorna un JPanel on hi ha tots els components de la interfície gràfica referents a la informació d'una compressió o descompressió individual.

VistaStatistics

Classe VistaStatisics
Autor: Ignasi Sant Albors
Mètodes públics:

- **VistaStatistics(IOUtils plOUtils, VistaPrincipal vp):** Constructora que inicialitza les els atributs de tipus IOUtils i VistaPrincipal
- **getPanelStatistics():** Mètode que retorna un JPanel on hi ha tots els components de la interfície gràfica referents a les estadístiques històriques.

VistaCompare

Classe Vista Compare
Autor: Ignasi Sant Albors
Mètodes públics:

- **VistaCompare(String[] s):** Constructora de la classe VistaCompare, rep com a paràmetres un String array amb el contingut del fitxer original i el contingut del fitxer tractat després de fer la compressió i descompressió.

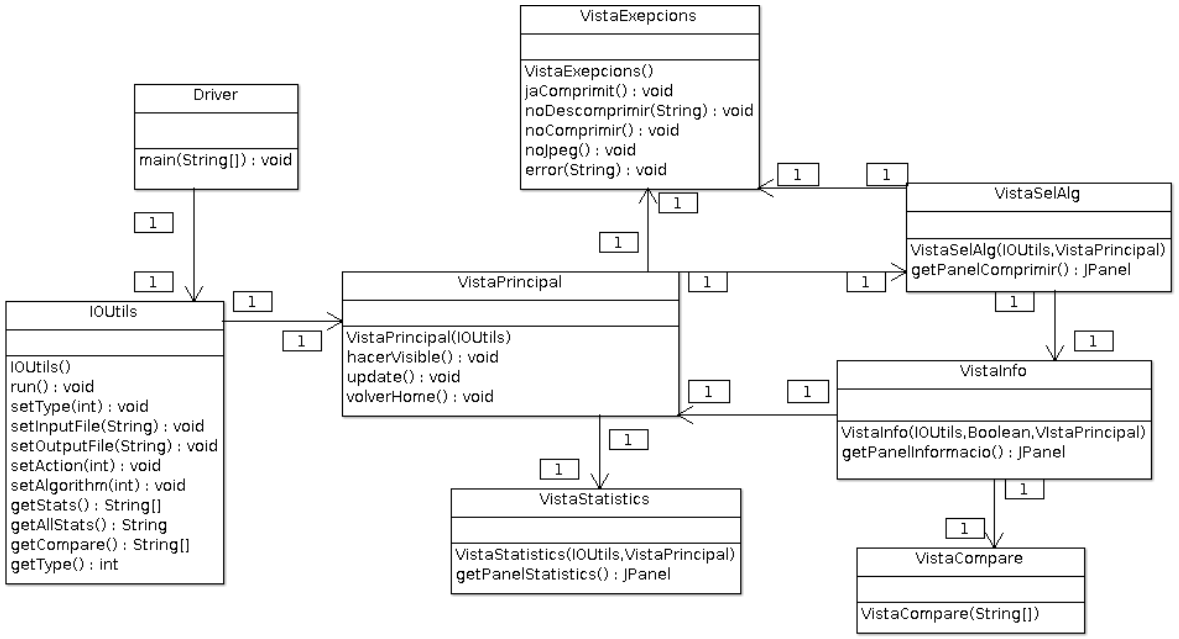
VistaExepcions

Classe VistaExepcions
Autor: Ignasi Sant Albors
Mètodes públics:

- **VistaExepcions():** Constructora
- **jaComprimit():** Mètode que fa aparèixer un pop-up d'error dient que l'arxiu ja està comprimit.
- **noDescomprimir(String extension):** Mètode que fa apareixer un pop-up d'error dient que no es pot descomprimir el fitxer.
- **comprimir():** Mètode que fa aparèixer un pop-up d'error dient que no es pot comprimir el fitxer.
- **noJpeg():** Mètode que fa aparèixer un pop-up d'error dient que el fitxer no pot ser comprimit amb l'algoritme JPEG
- **error(String message):** Mètode que fa aparèixer un pop-up d'error dient que hi ha hagut un error inesperat, indicant quin és aquest error

Diagrama de classes Capa de la Presentació

Aquest diagrama de classes té en compte totes les classes de la capa de presentació. Com es veu en la imatge totes les relacions son 1 a 1. També es veu que IOUtils (controlador de presentació) es comunica amb la classe VistaPrincipal i aquesta s'encarrega de tota la Interfície Gràfica amb l'ajuda de totes les altres vistes.



Capa de Domini.

Aquesta capa conté les classes del domini. Entre elles trobem els controladors de la capa, els algoritmes i les excepcions que usen aquestes classes.

Algoritmes

En aquesta secció es documenten els algoritmes que hem implementat en la pràctica. **Tots els algoritmes hereden de la classe Algorithm** explicada a continuació.

Algorithm

Classe Algorithm
Autor: Jordi Garcia Aguilar.
Mètodes públics (obviant getters i setters i constructora estàndard): És classe abstracta: Té els següents mètodes abstractes:

- **compress():** Aplica l'algoritme de compressió a l'input.
  - **Pre:** S'ha cridat prèviament al mètode *setData(d)*.
  - **Post:** El String resultant a aplicar l'algoritme de compressió a l'String *d*.
- **decompress():** Aplica l'algoritme de descompressió a l'input.
  - **Pre:** S'ha cridat prèviament al mètode *setData(d)*.
  - **Post:** L'String resultant a aplicar l'algoritme de descompressió a l'String *d*.

- **getData():** Retorna les dades d'entrada per al algoritme proveïdes prèviament per la funció *setData(d)* o, d'altre manera retorna null.
- **setData(String data):** Mètode per proveir de les dades d'entrada per els algorismes de compressió i/o descompressió.
- **getId():** Retorna un valor de tipus *int* que és l'identificador numèric de l'algoritme
- **getExtension():** Retorna un String que conté l'extensió que identifica els fitxers comprimits amb l'algoritme

Jpeg

Classe JPEG, hereda d'Algorithm
Autor: Joaquim Ferrer Sagarra
Mètodes Públics: No en té.

Explicació de l'algoritme i la seva implementació:

L'algoritme JPEG està implementat per mitjà de la classe abstracta JPEG. Es va decidir fer abstracta perquè hi ha moltes estructures de dades que són compartides. Aquesta classe té com a atributs destacables les dues matrius de quantització i un objecte de la classe *Imatge*. La resta d'atributs són explicats més endavant. Cal destacar que, per a la quantització s'han

usat matrius per poder-se adaptar al format que tindran les matrius que es generaran al llarg de l'algoritme.
El màxim cost asimptòtic es troba en la creació de l'arbre de Huffman, aquest cost és de O(n*(logn)^2) (veure documentació de la classe HuffmanTree per veure la justificació dels costos assimptotics de l'algoritme, compressió punt c).

Compressor JPEG

Classe JPEGCompressor, hereda de JPEG
Autor: Joaquim Ferrer Sagarra
Mètodes Públics (obviant getters i setters i constructora estàndard):

- **compress():** Mètode sobreescrit de la classe Algorithm.
    - **Pre:** L'atribut *Data* no es null.
    - **Post:** Retorna un fitxer comprimit amb l'algoritme JPEG.

Explicació de l'algoritme i la seva implementació:

Per realitzar una compressió, el primer pas és instanciar un objecte de la classe **JPEGCompressor.** Al fer la instància, la constructora inicialitza l'atribut *imatge* de la classe *Imatge* amb un objecte buit de la classe *Imatge*. Seguidament, cal cridar al mètode públic **setData()** per donar valor a l'atribut *data*. Aquest atribut és un **String** que conté la imatge en format PPM**.**
El següent pas és cridar al mètode públic **compress().** Aquest mètode realitzarà la compressió amb els següents passos:

1. Cridar a la classe **setData()** per introduir la imatge a comprimir. Aquesta imatge serà rebuda en format String. Un cop rebuda, la funció cridarà a la *Imatge.creaImatgeDePPM()* que prepararà l'atribut *imatge* per a la conversió (veure documentació de la classe imatge).
2. Cridar a la funció *operaToYCrCb()* de l'atribut imatge que convertirà la *imatge* de l'espectre de colors RGB a YCbCr.
3. Cridar a la funció privada *transformBlocks*() que s'encarregarà de la creació i les operacions a realitzar amb les **matrius** a partir de la imatge, es detalla la funció en els següents subpunts:
    a. Crear tres **arrays de matrius** (un array per la il·luminància, un per la crominància blava i un per la crominància vermella) que serviran per contenir cada un d'ells totes les matrius de 8x8 de la fotografía. L'algoritme té en compte que la fotografia pot no ser divisible entre matrius de 8x8. En aquest cas, estendrà l'últim pixel que ha vist per acabar de completar la fotografia i fer-la divisible en **matrius de 8x8.**
    b. Aplicar per cada matriu la transformació DCT2
4. Dividir els elements de les matrius obtingudes entre les matrius de quantització definides a la superclasse.
5. Es recorren aquestes matrius en zigzag guardant els elements en **tres vectors**. Un per la il·luminància, un per la crominància blava i un per la crominància vermella. Per fer-ho s'usarà una classe estàtica anomenada *creazigzag().* Aquesta classe rep com a paràmetre una matriu i retorna un vector corresponent a recorrer aquesta matriu en

ZigZag. Després de cridar aquesta funció per cada matriu inclosa en els **arrays de matrius** de il·luminància i crominància es genera un vector amb tots els valors de totes les matrius (primer la il·luminància, després les dues crominàncies). S'usa un **vector** per la facilitat que presenta a l'hora de ser recorregut. A més, gràcies a les funcions que ens dona la classe *Vector* de java podem tractar i editar el seu contingut de forma molt més senzilla que amb un array. Aquest pas estava inclòs en l'algoritme per després fer més senzill l'ús de huffman. Malauradament, l'algoritme de huffman no està implementat com l'estàndard de JPEG així que aquesta part del codi no és molt útil. Tot i això, com que és feina que s'ha realitzat i per ser coherents amb l'algoritme JPEG s'ha decidit deixar.

6. S'envia aquest llarg vector a la classe *HuffmanTree* que per mitjà de la seva funció *encode()* (veure documentació de la classe *HuffmanTree*) ens retornarà una string resultant de fer la compressió del vector amb l'algoritme de Huffman.
7. Es crea una capçalera que inclou en ascii l'amplada i l'alçada de la imatge resultant. Aquesta capçalera més l'string retornat de la compressió de huffman és retornat per la funció. Els controladors de la capa de domini i presentació s'encarregaran de que aquest String esdevingui un fitxer amb extensió *.jpeg*.

Descompressor JPEG

Classe JPEGDecompressor, hereda de JPEG
Autor: Joaquim Ferrer Sagarra
Mètodes Públics (obviant getters i setters i constructora estàndard):

- **setData():** Mètode sobreescrit de la classe Algorithm.
  - **Pre:** Rep un string que ha sigut generat per la classe *JPEGCompressor.*
  - **Post:** L'atribut *imatge* està preparat per dur a terme la descompressió correctament.
- **decompress():** Mètode sobreescrit de la classe Algorithm.
  - **pre:** L'atribut imatge no és null.
  - **post:** Retorna una string que codifica una imatge en PPM fruit de la descompressió.

Explicació de l'algoritme i la seva implementació:

Per a realitzar una descompressió, el primer pas és instanciar un objecte de la classe **JPEGDecompressor.** Al fer la instància, la constructora inicialitza l'atribut *Imatge* de la classe *Imatge.* Seguidament, cal cridar al mètode **setData()** per donar valor a l'atribut data. Aquesta crida prepara l'atribut *imatge* de la classe *Imatge* donant-li els valors necessaris per a realitzar la descompressió per mitjà de la crida al mètode *Imatge.retallaHeaders()* (veure documentació de la classe *Imatge*). També, per mitjà de la classe HuffmanTree descomprimeix la part de l'String rebut que codifica la imatge guardant-ho en **tres vectors.** En un hi haurà totes les il·luminànices, en altre les Crominàncies blaves i en el tercer les crominàncies vermelles. S'usen tres vectors perquè s'han de poder recórrer de forma senzilla i la classe Vector de java ens dóna aquesta possibilitat. Seguidament cal cridar al mètode **decompress()** que realitzarà la descompressió seguint els següents passos:

1. Inicialitzar les matrius necessàries per a la descompressió. Del mètode *setData()* s'obtenen **tres vectors**: un d'il·luminància, un de crominància blava i un de crominància vermella. Cadascun d'aquests conté el resultat d'haver recorregut en zigzag els arrays de matrius de 8x8 creats pel la compressió (veure punt 5 de l'explicació de l'algoritme JPEGCompressor). Doncs, en aquest punt de la descompressió cal desfer aquest zigzag tornant a muntar els **tres arrays de matrius**.
2. Multiplicar cada element de cada matriu de cada array de matrius pel seu corresponent factor a de la taula de quantització.
3. Cridar a la funció privada *transformBlocks*() que s'encarregarà d'aplicar la transformació DCT3 per obtenir les **matrius 8x8** en les que es basarà la construcció de la fotografia.
4. Es recorren aquestes matrius obtingudes després de la descompressió en vista a crear un **vector de Colors** (veure documentació de la classe imatge) que és passat a la classe *Imatge* per mitjà del seu mètode *setImatge().* Ara, l'atribut *imatge* es troba en el mateix estat que al finalitzar el punt 2 de la compressió (veure documentació de la classe JPEGCompressor).
5. Es crida al mètode públic *operaFromYCbCr* de l'atribut *imatge* de la classe *Imatge* per transformar-lo de l'espectre de color YCbCr a RGB.
6. Per mitjà del mètode de l'atribut *imatge* de la classe *Imatge creaImatgeFinal()* s'afegeix una capçalera de PPM i es codifica tota la imatge en un format correcte perquè pugui ser interpretat com a PPM. Això retorna a mode d'String que es retorna al controlador de la capa de domini perquè pugui ser convertit en un fitxer PPM posteriorment.

LZW

Classe LZW, hereda d'Algorithm
Autor: Ignasi Sant Albors
Mètodes Públics:

- **getId():** Mètode que retorna l'identificador(int) de l'algoritme.
- **getExtension():** Mètode que retorna l'extensió(String) de l'algoritme.
- **getData():** Mètode que retorna el String a tractar.
- **setData(String data):** Mètode per establir el String que s'ha de tractar.
- **compress():**
    - **pre:** S'ha introduit un String que es vol comprimir
    - **post:** S'ha comprimit el String introduit i es retorna el resultat en forma de String.
- **decompress():**
    - **pre:** S'ha introduit un String que es vol descomprimir
    - **post:** S'ha descomprimit el String introduït i es retorna el resultat en forma de String.

Explicació de l'algoritme i la seva implementació:

L'algorisme LZW és un algorisme de compressió sense pèrdua (lossy-less). El punt fort d'aquest respecte altres algorismes de compressió és que pot crear sobre la marxa, de manera automàtica i en una sola passada, un Hashmap de cadenes que es troben dins del text a comprimir mentre, al mateix temps, es procedeix a la seva codificació. Aquest diccionari no s'inclou en el fitxer comprimit ja que el propi descompressor el pot reconstruir usant la mateixa lògica amb que ho fa el compressor i, si està ben codificat, obtindrà exactament les mateixes cadenes de caràcters que el Hashmap del compresor tenia.
Es comença introduint inicialment al Hashmap 256 entrades (de la 0 a la 255), cadascuna per un caràcter (byte) possible. A aquesta taula se li van afegint caràcters consecutius que es llegeixen encara que no sigui possible preveure si aquest codi serà reutilitzat més endevant.
En aquest detall és on es troba la màgia de l'algorisme, ja que al construir el Hashmap sobre la marxa només cal fer una passada sobre el text, i com que la regla de construcció del diccionari és tan simple, el descompresor el pot reconstruir a partir del text comprimir mentres el llegeix, evitant així incloure'l dins del text comprimit. Es pot objectar que el diccionari estarà ple de codis que no s'utilitzaran i per tant serà innecessàriament gran, però en la pràctica el diccionari no creix massa i encara si ho fes no importa molt ja que l'objectiu és que l'arxiu comprimit sigui petit tot i que els processos de compressió i descompressió poguessin ocupar molta memòria amb el diccionari.
Les entrades de l'diccionari poden representar seqüències de caràcters simples o seqüències de codis de tal manera que un codi pot representar dos caràcters o pot representar seqüències d'altres codis prèviament carregats que al seu torn representin, cada un d'ells, altres codis o caràcters simples, o sigui que un codi pot representar des d'un caràcter a un nombre indeterminat de caràcters. En realitat, l'algoritme no discrimina entre codis i caràcters simples doncs el Hashmap es carrega inicialment de codis que representen els primers 256 caràcters simples de manera que aquests no són més que altres codis dins el mateix diccionari.
Cada vegada que es llegeix un nou caràcter es revisa el diccionari per veure si forma part d'alguna entrada prèvia. Tots els caràcters estan inicialment predefinits dins del diccionari així que sempre hi haurà com a mínim una coincidència, però, el que es busca és la cadena més llarga possible. Si el caràcter llegit no forma part de més d'una cadena més llarga, llavors s'emet la més llarga que s'hagués trobat i s'afegeix a l'diccionari una entrada formada per qualsevol que hagués estat el codi previ i aquest nou codi. Si el caràcter llegit si forma part de més d'una cadena de el diccionari, es llegeix un nou caràcter per veure si la seqüència formada pel caràcter previ i el nou és alguna de les trobades al diccionari. Mentre els caràcters successius que es vagin llegint ofereixin més d'una entrada possible al diccionari, se segueixen llegint caràcters. Quan la cadena només té una entrada al diccionari, llavors s'emet el codi corresponent a aquesta entrada s'incorpora a l'diccionari una nova entrada que representa l'últim codi emès i el nou.

Pseudo codi LZW

- **Compressió:**

```
STRING = getData(); //obtenir input
WHILE hi hagin caràcters a l'input DO
   CHARACTER = obté caràcter input
   IF STRING+CHARACTER està al diccionari String llavors
      STRING = STRING+CHARACTER
   ELSE
      codi output per  STRING
      afegeix STRING+CHARACTER al diccionari String
      STRING = CHARACTER
   FI de IF
FI de WHILE
codi output per STRING
```

- **Descompressió:**

```
llegeix CODI_ANTIC
output CODI_ANTIC
WHILE hi hagin caràcters a l'input DO
   llegeix CODI_NOU
   STRING = obtenir traducció CODI_NOU
   output STRING
   CHARACTER = primer caràcter de STRING
   afegeix CODI_ANTIC + CHARACTER a la taula de traduccions
   CODI_ANTIC = CODI_NOU
FI de WHILE
```

Estructures usades i justificació:

- **HashMap<String, Integer> taula:** Aquesta estructura de dades és la més adequada per a la compressió en LZW ja que es vol assignar un codi numèric a un conjunt únic i irrepetible de caràcters. És per això que la clau serà un String i el valor un Integer, d'aquesta manera el HashMap no permetrà tenir claus String duplicades.Per altre banda si que permetrà tenir dos valors iguals associats a les claus però es farà que mai es doni aquesta situació ja que ens interessa obtenir una assignació 1 a 1. Com que tampoc volem mantenir l'estructura ordenada el HashMap ja ens va bé. El cost amortitzat d'inserció i consulta es O(1) assumint que la funció has funciona correctament, per tant és l'estructura més adequada per al problema que ens ocupa. Com que al fer la compressió el tamany de l'estructura usada pot ser molt gran ja va bé

l'ús del HashMap, ja que si aquest s'omple molt s'autoexpandeix i tots els element es tornen a colocar automàticament tenint en compte la funció de hash.

- **HashMap<Integer, String> taula:** Aquesta estrucutra de dades s'adequa perfectament a la descompressio en LZW ja que en aquest cas s'ha d'asignar una cadena de caràcters a un valor numèric únic. Per fer-ho, la millor opció és usar un HashMap ja que no es volen claus numèriques repetides , no interessa mantenir l'estructura ordenada , es vol un cost minim d'inserció i consulta i es necessita una estructura que s'actualitzi automàticament a mesura que s'omple.
- **Vector <Integer> op:** S'ha usat aquesta estructura en la descompressió en LZW ja que es necessitava guardar el conjunt de codis numèrics que s'han de descomprimir en forma de Integer, ja que arribaven en un String[]. A part el cost d'inserció i consulta es O(1). Per tant aquesta és l'estructura de dades més adequada per tractar el problema.
- **String[] parts:** Conté els codis que representen la compressió en forma de String. S'usa aquesta estructura per extreure tots els codis que es reben, ja que aquets venen en un String en que cada codi està separat per ":".

LZ78

Classe LZ78, hereda d'Algorithm
Autor: Jordi Garcia Aguilar
Mètodes Públics:

- **getId():** Mètode que retorna l'identificador (int) de l'algoritme.
- **getExtension():** Mètode que retorna l'extensió(String) de l'algoritme.
- **getData():** Mètode que retorna les dades d'entrada.
- **setData(String data):** Mètode per proveïr les dades d'entrada a la instancia.
- **compress():** Mètode que aplica l'algoritme de compressió de LZ78 a les dades d'entrada
    - **Pre:** S'ha cridat a *setData()* prèviament.
    - **post:** Retorna un String que es el resultat d'aplicar a les dades d'entrada l'algoritme de compressió LZ78

- **decompress():** Mètode que aplica l'algoritme de descompressió de LZ78 a les dades d'entrada
    - **Pre:** S'ha cridat a *setData()* prèviament amb unes dades comprimides amb LZ78.
    - **Post:** Retorna un String que es el resultat d'aplicar a les dades d'entrada l'algoritme de descompressió LZ78.

Explicació de l'algoritme i la seva implementació:

L'algorisme LZ78 és un algorisme de compressió sense pèrdua (lossy-less). Aquest algoritme és més eficient en quant més llarg sigui el text a comprimir. Això és degut a que l'algoritme intenta evitar seqüències repetides.

Per a comprimir es recorre el text caracter a caracter. Per a cada iteració es llegeix un caràcter i es comprova si existeix alguna entrada en el diccionari per aquest caràcter. En cas de que no existeix cap entrada, s'afegeix al diccionari el caràcter i s'afegeix al resultat el index de l'últim caràcter si aquest sí que hi era al diccionari o un 0 en el seu defecte i a continuació el caràcter en qüestió. En el cas de que sí que es trobi en el diccionari es continua a la següent iteració.

Estructures usades i justificació:

- **HashMap<String, Integer> map:** És l'estructura de dades principal en una de les funcions més importants de la clase: compress(). En aquesta funció calia anar comprovar per cada iteració del bucle que recorre el fitxer a comprimir si cal guardar un String o no en una estructura de dades. En una primera implementació s'usaba un ArrayList i el seu mètode indexOf() per fer la comprovació. Aquest mètode té un cost de O(n) i provocaba per tant que la funció sencera tingues un cost quadràtic. És per aquest motiu que finalment es va implementar el mètode utilitzant el Hashmap ja que tant el mètode containskey(), usat per saber si cal afegir o no el caràcter corresponent a la iteració del bucle que recorre el text, com el mètode put(), usat per afegir l'element en cas de que no s'hagi fet prèviament, tenen cost constant. D'aquesta manera utilitzar reduir el cost de la funció compress() a O(n).

Controlador

En aquesta secció es detalla la classe que actua com a controlador de la capa de domini.

Fitxer

Classe Fitxer
Autor: Jordi Garcia Aguilar
Mètodes Públics:

- **Fitxer():** Constructora. Instancia la classe Compressio i Descompressio i inicialitza el controlador de la capa de dades.

- **compress():** Mètode que inicialitza l'algoritme a *Compressio* i crida el mètode de comprimir d'aquest .
    - **Pre:** Ruta absoluta del fitxer d'entrada (String), ruta absoluta del director de sortida (String), tipu de compressió (int: 0 -> compressió de fitxer, 1 -> compressió), identificador de l'algoritme a utilitzar(int: 0 -> LZ78, 1 -> LZW, 2 -> JPEG)
    - **Post:** El fitxer d'entrada s'ha comprimit amb el nom original afegint la extensió corresponent l'algoritme utilitzat. El fitxer comprimit es trobarà en: o bé la ruta especificada (en cas que no sigui un String buit), o bé en el directori on es troba el fitxer d'entrada. Retorna un array d'Strings amb els valors estadistics de la compressió, per ordre en l'array:
        - Nom del fitxer d'entrada
        - Identificador de l'algoritme utilitzat
        - Mida del fitxer d'entrada
        - Mida del fitxer comprimit
        - Grau de compressió
        - Duració de compressió (ms)

- **decompress():** Mètode que inicialitza l'algoritme a *Descompressio* i crida el mètode de descomprimir d'aquest .
    - **Pre:** Ruta absoluta del fitxer d'entrada (String), ruta absoluta del director de sortida (String), identificador de l'algoritme a utilitzar(int: 0 -> LZ78, 1 -> LZW, 2 -> JPEG)
    - **Post:** El fitxer d'entrada s'ha descomprimit amb el nom original. El fitxer descomprimit es trobarà en: o bé la ruta especificada (en cas que no sigui un String buit), o bé en el directori on es troba el fitxer d'entrada. Retorna un array d'Strings amb els valors estadistics de la descompressió, per ordre en l'array:
        - Nom del fitxer d'entrada
        - Identificador de l'algoritme utilitzat
        - Mida del fitxer d'entrada
        - Mida del fitxer descomprimit
        - Grau de descompressió
        - Duració de la descompressió (ms)

- **llegirFitxer():** Mètode que demana al controlador de la capa de dades el contingut d'un fitxer
    - **Pre:** Ruta absoluta del fitxer a llegir
    - **Post:** String amb el contingut del fitxer
- **writeToFile():** Mètode que demana al controlador de la capa de dades escriure en un fitxer
    - **Pre:** Contingut a escriure al fitxer (String), ruta absoluta al fitxer (String)
    - **Post:** S'ha escrit el contingut al fitxer
- **llegirDescomp():** Mètode que demana al controlador de la capa de dades el contingut d'un fitxer i el tracta per a ser descomprimit

    - **Pre:** Ruta absoluta al fitxer (String)
    - **Post:** Un array de Strings que conté, en ordre:
        - L'identificador de l'algorisme amb el que ha estat comprimit
        - Nom del fitxer original (abans de ser comprimit)
        - Contingut de compressió
- **saveStatistic():** Mètode que demana al controlador de la capa de dades afegir una nova entrada a l'històric d'estadístiques.
    - **Pre:** Ruta absoluta del fitxer comprimit o descomprimit (String), identificador de l'algoritme emprat (int), grau de compressió/descompressió (double), duració de la compressió/descompressió en milisegons (long).
    - **Post:** L'històric conté una entrada més amb els nous valors.

- **getStats():** Mètode que demana al controlador de la capa de dades el contingut del històric d'estadístiques de les compressions/descompressions realitzades.
    - **Post:** Un String amb el contingut del fitxer.

Excepcions

PPM mal format

Classe PPMBadFormated
Excepció que es llança quan la classe imatge detecta que:

- La imatge PPM introduida no és un PPM P6
- La imatge PPM no té un nombre màxim de color 255 (el color no es codifica amb 24 bits)
- El binari de colors de la imatge no és igual a l'amplada per l'alçada per 3.

PPM massa gran.

Classe PPMTooBig
Excepció que es llança quan la classe imatge detecta que la imatge que s'ha introduit és massa gran. Degut a l'alt nombre les funcions recursives que monten l'arbre de Huffmann, una imatge d'aquest tamany pot sobresaturar la pila i no es podrà comprimir. Per salvar això caldria refer tot l'algoritme de HuffmanTree (veure documentació de la classe HuffmanTree) perquè funcionés amb funcions iteratives en comptes de funcions recursives.

Altres Classes del domini

Imatge

Classe Imatge
Autor: Joaquim Ferrer Sagarra
Mètodes Públics (obviant getters i setters i constructora estàndard):

- **operaFromYCbCr():**

    - **Pre:** L'atribut *imatge* de la classe és un **vector de colors en RGB** (veure documentació de la classe Color).
    - **Post:** L'atribut imatge de la classe és un **vector de colors en YCbCr.**
- **operaToYCbCr():**
    - **pre:** L'atribut imatge de la classe és un **vector de colors en YCbCr.**
    - **post:** L'atribut *imatge* de la classe és un **vector de colors en RGB** (veure documentació de la classe Color)
- **crealmatgeDePPM(String):**
    - **Pre:** L'String passat per paràmetre és una imatge PPM P6 amb un color codificat amb 24 bits.
    - **Post:** L'atribut *imatge* està preparat per dur a terme la compressió correctament. S'ha donat valor als atributs *Imatge*, *alt, ample MaxVal* (nombre màxim amb el que es pot codificar un color, ha de ser 255) i *magicNum* (Nombre màgic del PPM, ha de ser P6).
- **crealmatgeFinal():**
    - **pre:** L'atribut imatge és un **vector de colors d'RGB.**
    - **post:** Retorna una string que codifica una imatge en PPM amb la seva capçalera i les seves dades en binari.
- **getColorPerIndex(int):**
    - **pre:** El paràmetre és un índex de l'atribut *imatge.*
    - **post:** es retorna un objecte de la classe *Color* corresponent a l'element que hi havia a la posició passada per paràmetre de l'atribut *imatge*.

Explicació de la classe i la seva implementació:

La idea principal d'aquesta classe és permetre tractar d'una forma senzilla una imatge en PPM, tant sigui per poder, des d'aquesta imatge crear una estructura de dades operable en java com per , des d'aquesta estructura de dades operable crear una imatge PPM. Doncs, el punt essencial d'aquesta classe es que pugui funcionar a mode d'estructura de dades que permeti un tracte senzill i aïllat de la lògica de l'algoritme amb una imatge PPM.
L'implementació d'aquesta estructura es fa per mitjà d'un **Vector d'elements de la classe Color** (veure documentació de la classe color). S'ha escollit un vector perquè una imatge PPM és bàsicament un vector. L'avantatge que presenta un vector davant d'una matriu és que és més fàcilment iterable per mitjà de la construcció *for each*. Doncs, es poden recórrer els seus elements si omplir-los des del binari PPM sense haver de fer contadors externs ja que els bits que codifiquen els colors dins d'un PPM estan disposats a mode de Vector. A partir d'aquest vector, recorre'l i construir les matrius de 8x8 presenta la mateixa dificultat que tindria si s'hagués usat una matriu per a codificat la imatge doncs, clarament l'estructura que adequada per iementar aquesta classe era el **Vector.**
Aquest vector conté tripletes de la classe color. Això fa que el seu tracte dins de l'algoritme de compressió i descompressió sigui molt natural. Gràcies a aquesta estructura el jpeg pot referir-se a colors en posicions concretes de la imatge, obtenir qualsevol de les tres components d'aquest color, transformar tots els colors del vectors de RGB a YCbCr i viceversa, crear l'estructura a partir d'un PPM i crear un PPM a partir de l'estructura.
Un altre dels punts més importants de la classe són **els atributs amplada i alçada** de la imatge. La classe JPEG necessita rebre la informació constantment i la classe imatge li

proporcionarà sempre tant en la compressió com en la descompressió ja sempre, en l'inici d'aquestes dues funcionalitats el primer que es fa és crear un atribut d'aquesta classe i donar el valor correcte *amplada* i *alçada* (veure documentació de les classes JPEGCompressor i JPEGDecompressor). La classe *Imatge* és un bon exemple de les tècniques de programació orientada a objectes ja que permet aïllar les operacions que respecten a la imatge de les que formen part de l'algoritme. Si es decidís canviar JPEG per un altre algoritme, la classe *Imatge* seguiria proporcionant una interfície còmode, vàlida i útil per a poder tractar amb imatges de tipus PPM P6.

Color

Classe Color
Autor: Joaquim Ferrer Sagarra
Mètodes Públics (Obviant getters i setters):

- **Color (int,int.int):**
  - **pre:** S'introdueixen tres valors enters del 0 al 255 que codifiquen un color en RGB o YCbCr
  - **post:** S'ha creat un objecte de la classe color amb els valors introduïts per paràmetre.

Explicació de la classe i la seva implementació:

Aquesta classe codifica amb tres enters una tripleta RGB o YCbCr. Està pensada per tenir un tracte fàcil amb els píxels d'una imatge i seguint les dinàmiques de la programació orientada a objectes.

HuffmanTree

Classe JPEGDecompressor
Autor: Joaquim Ferrer Sagarra
Mètodes Públics (obviant getters i setters i constructora estàndard):

- **encode(Vector<Integer>):**
  - **Pre:** Rep un Vector d'enters.
  - **Post:** Retorna un String fruit de la compressió per l'algoritme de Huffman.
- **decode(String):**
  - **pre:** Rep un String que ha sigut generat amb la funció encode d'aquesta classe.
  - **post:** Retorna un **vector d'*Integer*** fruit de la descompressió usant el mètode de Huffman de l'string donada.

Explicació de la classe i la seva implementació:

Aquesta classe ha sigut creada amb la intenció d'aillar la compressió pel mètode de Huffman de l'algoritme JPEG. Això permet poder canviar el mètode de compressió (ara

Huffman) per un altre sense haver de modificar el codi de l'algoritme JPEG, funció molt útil per fer debugging per mitjà de drivers i stubs.
Aquesta classe té com a base de tota lògica un **arbre implementat amb un conjunt d'elements de la classe privada *Node.*** El node arrel d'aquest arbre és guardat en un atribut anomenat *root.* L'accés a aquest atribut permet recórrer l'arbre de forma recursiva i poder generar un diccionari de traducció a codi de Huffman. Les seves principals funcionalitats són compressió i descompressió i es detallen a continuació:

- **Compressió:** Es fa per mitjà de la funció encode i té els següents passos:
    a. Es crea un **HashMap que relaciona cada enter del vector d'input (clau) amb la seva freqüència d'aparició (valor).** Es recorre el vector d'enters passat com a input. Si l'enter trobat no s'ha vist abans s'inclou al HashMap amb valor 1. Si s'ha vist abans es suma 1 al valor que conté. Un cop recorregut tot el vector, el HashMap conté un diccionari que relaciona cada enter vist amb la seva freqüència d'aparició al vector. S'ha decidit usar un **HashMap perquè en cas mitjà el cost d'inserció és O(1), sent O(n) el cas pitjor,** cas que és complicat si la funció de hash està ben implementada. Doncs, compto que la majoria dels meus accessos al HashMap es faran en temps O(1) o molt pròxim a O(1).
    b. Es recorre aquest HashMap element a element (cost O(nombre d'elements del HashMap)) i **es crea un element de la classe Node** amb cadascun d'ells. Aquest *node* conté el valor, la freqüència d'aparició i els nodes fills dret i esquerra en l'arbre. Al final hi haurà un node per cada enter diferent aparegut al vector paràmetre de la funció. Aquest node creat s'insereix en una **cua de prioritat inversa en funció de la freqüència d'aparició de l'enter**. S'usa aquesta cua perquè és necessari tenir ordenats els elements de menor a major grau d'aparició per a crear un arbre de Huffman. El cost d'inserció a una Cua de prioritat és de (On*logn) sent *n* el nombre d'enters diferents. Doncs, la complexitat de la creació d'aquesta cua és d'O(n*logn).
    c. Es genera un arbre de Huffman a partir d'aquesta cua de prioritats. Això es fa per mitjà de l'algoritme de Huffman, és a dir, creant un *MinHeap* en funció de el nombre d'aparicions de cada enter en el vector d'input. El cost d'aquesta part del codi és de O(n*(logn)^2) sent *n* el nombre d'enters diferents. Això és per la dinàmica de creació de l'arbre. Cal pensar que el funcionament és: obtenir els dos nodes superiors de la cua O(1), crear un node pare per aquests dos nodes obtinguts i inserir-ho a la cua O(n*logn), repetir aquest procés fins a crear un arbre de Huffman: O(n*logn*logn).
    d. Es crea un **HashMap** i es recorre l'arbre recursivament O(logn). Si es va a l'esquerra, es codifica el pas amb un 0, si es va a la dreta amb un 1. Cada cop que s'arriba a una fulla s'inserta l'element al **HashMap** O(1)**.** Un cop recorregut l'arbre s'ha obtingut un diccionari que codifica per cada element la seva representació per l'arbre de huffman creat. També s'ha creat un **String** que servirà posteriorment per a guardar l'arbre de Huffman al resultat final. Per crear aquest String, cada cop que s'arriba a un node que **no** és fulla s'insereix un 0. Quan s'arriba a una fulla s'insereix un 0 i 16 bits que codifiquen el valor de l'enter que hi ha en aquesta fulla (no es guarden les freqüències perquè no seran necessàries per a la descompressió).

e. Es recorre el vector paràmetre O(n) sent *n* el nombre d'elements del vector passat per paràmetre codificant en un **String d'1s i 0s** l'equivalent al vector després d'aplicar una codificació usant l'arbre de Huffman creat.
f. Es crea un string ajuntant: La llargada del codi de Huffman + el codi de Huffman + l'arbre codificat al punt d.
g. Es recorre aquest string aplicant shifts i ors lògiques per a convertir-la a un string en binari. És retronat per la funció.

- **Descompressió:** Es fa per mitjà de la funció decode i té els següents passos.
    a. Es parteix d'un **String** passat com a paràmetre que conté primer 32 bits que codifiquen la llargada del codi de huffman, seguidament hi ha el codi de Huffman de la imatge i posteriorment l'arbre codificat al punt d de la compressió. Aquests strings tenen un format binari que es converteix a un format **String d'1s i 0s** aplicant una funció inversa a la del punt g de la compressió. Es separen aquests tres camps.
    b. Es recorre el vector amb una funció iterativa que montarà l'arbre de huffman a partir del codi generat al punt d de la compressió. El cost d'aquesta funció és O(tamany de la codificació de l'arbre);
    c. Recorrer String del codi de Huffman de la següent manera: Fins que troba una fulla, si hi ha un 0 avança l'arbre a l'esquerra, si hi ha un 1 avança l'arbre a la dreta. Quan troba una full insereix en un **vector d'enters** el valor de l'enter del node i torna a l'arrel. Un cop recorregut tot l'string s'ha obtingut un vector d'enters que és retornat com a resultat de la descompressió.

Node

Aquesta classe permet crear parelles de valors Integer-Integer o Integer-char. És utilitzada per implementar els algoritmes de compressió/descompressió LZ78

Classe Node

Autor: Jordi Garcia Aguilar

Mètodes Públics:

- **Node(Integer, Integer):** Constructora que retorna una instancia de Integer-Integer.
- **Node(Integer, char):** Constructora que retorna una instancia de Integer-char.
- **getIndex():**
    - **Pre:** És una instància Integer-Integer
    - **Post:** Retorna el primer paràmetre passat a la constructora
- **getAnterior():**
    - **Pre:** -
    - **Pro:** Retorna un Integer. Si la instància és Integer-Integer retorna el segon paràmetre passat a la constructora, si és Integer-char retorna el primer paràmetre.
- **getAnterior():**
    - **Pre:** És una instància Integer-Integer
    - **Post:** Retorna el segon paràmetre passat a la constructora.

Statistics

Classe Statistics, patró Singleton.

Autor: Jordi Garcia Aguilar
Mètodes Públics:

- **getStatistics():** Mètode per solicitar la instància de Statistics.
    - **Pre:** -
    - **Pro:** Retorna la instància de Statistics
- **initStats():** Mètode que inicialitza el contador que es farà servir per calcular les duracions de les compressions/descompressions.
- **saveStats:** Mètode que atura el comptador, calcula les dades per les estadístiques i afegeix una nova entrada a l'històric.
    - **Pre:** Ruta absoluta del fitxer comprimit/descomprimit (String), identificador numèric del algoritme emprat (int), mida original del fitxer (int), mida després d'aplicar l'algoritme (int).
    - **Post:** S'ha escrit una nova entrada en el històric d'estadistiques. Retorna un array d'Strings amb els valors estadistics de la compressió, per ordre en l'array:
        - Nom del fitxer d'entrada
        - Identificador de l'algoritme utilitzat
        - Mida del fitxer d'entrada
        - Mida del fitxer comprimit/descomprimit
        - Grau de compressió/descompressió
        - Duració de compressió/descompressió (ms)

Justificació del patró: La classe implementa el timer que mesura la durada de la compressió/descompressió. Ja que el sistema no permet multiple compressions o descompressions simultànies i per tal de assegurar la coherència del timer s'ha considerat que era adient aplicar el **patró *Singleton***.

Compressió

Classe Compressio
Autor: Jordi Garcia Aguilar
Mètodes Públics:

- **compress():** Mètode que gestiona la compressió d'un fitxer diferenciant si aquest és un directori o no .
    - **Pre:** Ruta absoluta del fitxer d'entrada (String), ruta absoluta del director de sortida (String), tipu de compressió (int: 0 -> compressió de fitxer, 1 -> compressió).
    - **Post:** El fitxer d'entrada s'ha comprimit amb el nom original afegint la extensió corresponent l'algoritme utilitzat. El fitxer comprimit es trobarà en: o bé la ruta especificada (en cas que no sigui un String buit), o bé en el directori on es troba el fitxer d'entrada. Retorna un array d'Strings amb els valors estadistics de la compressió, per ordre en l'array:
        - Nom del fitxer d'entrada
        - Identificador de l'algoritme utilitzat
        - Mida del fitxer d'entrada

        - Mida del fitxer comprimit
        - Grau de compressió
        - Duració de compressió (ms)
- **getAlgoData():** Mètode que retorna les dades d'entrada introduïdes a l'algoritme.
- **setAlgorithm():** Mètode que Instancia l'algoritme seleccionat a partir del seu identificador numèric

Explicació de la classe i la implementació

La classe Compressio gestiona les crides de compressió de fitxers. La funció més interessant d'aquesta classe es ***compressFolder()***, usada per comprimir directoris. A continuació s'explica la seva implementació.

Per a generar el fitxer comprimit Compressio utilitza un mètode del controlador de la capa de domini, getHierarchy(), el qual retorna el nom de tots els fitxers que conté el directori. Per cada fitxer es crida al mètode de compressió de l'algoritme seleccionat. El fitxer comprimit es el resultat de concatenar per a cada fitxer del directori a comprimir, el seu nom , la mida de la compressió i el contingut de la compressió.

Descompressió

Classe Descompressio
Autor: Jordi Garcia Aguilar
Mètodes Públics:

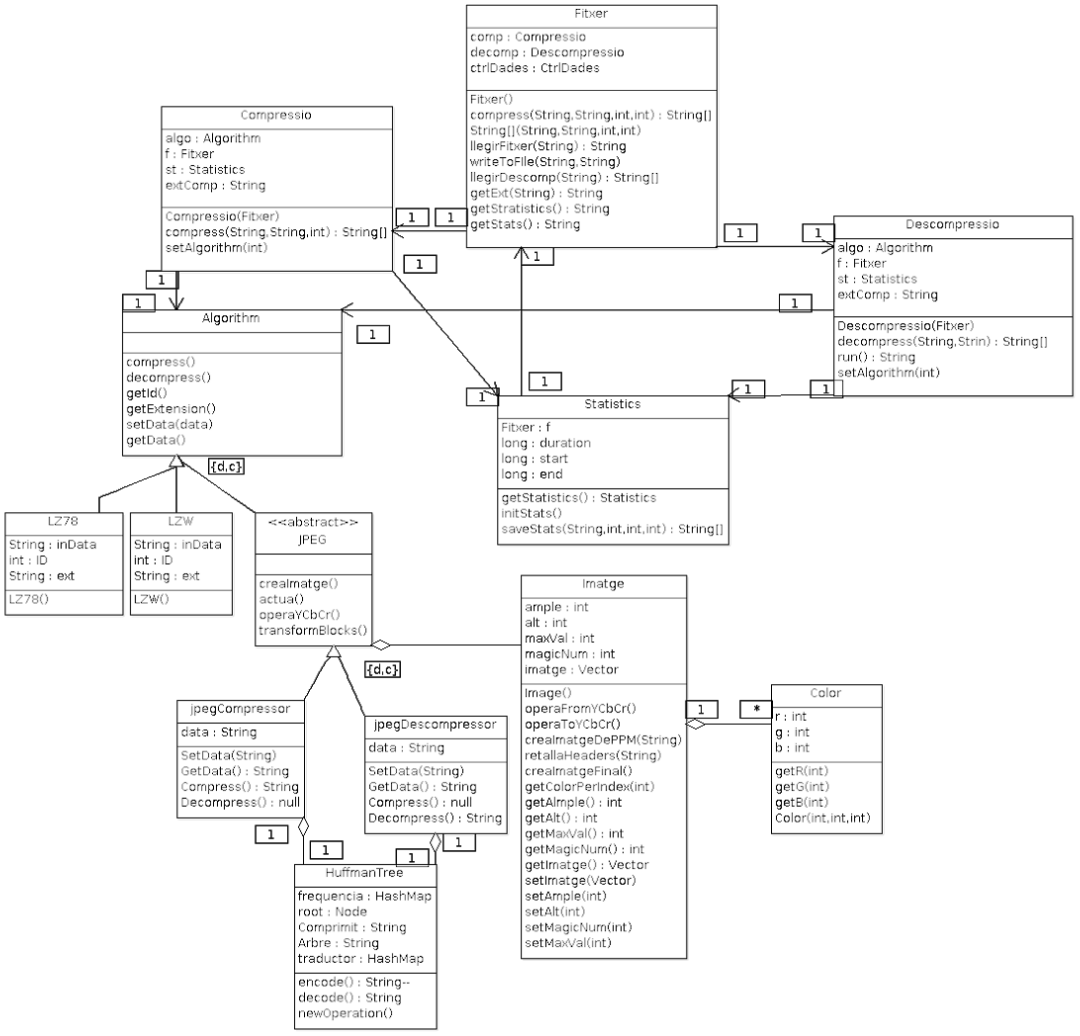
- **decompress():** Mètode que gestiona la compressió d'un fitxer diferenciant si aquest és un directori o no .
    - **Pre:** Ruta absoluta del fitxer d'entrada (String), ruta absoluta del director de sortida (String).
    - **Post:** El fitxer d'entrada s'ha descomprimit amb el nom original. El fitxer descomprimit es trobarà en: o bé la ruta especificada (en cas que no sigui un String buit), o bé en el directori on es troba el fitxer d'entrada. Retorna un array d'Strings amb els valors estadistics de la descompressió, per ordre en l'array:
        - Nom del fitxer d'entrada
        - Identificador de l'algoritme utilitzat
        - Mida del fitxer d'entrada
        - Mida del fitxer descomprimit
        - Grau de descompressió
        - Duració de la descompressió (ms)
    -
- **getAlgoData():** Mètode que retorna les dades d'entrada introduïdes a l'algoritme.
- **setAlgorithm():** Mètode que Instancia l'algoritme seleccionat a partir del seu identificador numèric
- **run():** Mètode que crida a la funció de descomprimir de l'algoritme instanciat prèviament amb *SetAlgorithm()*.
-

Explicació de la descompressió de directoris

Si el fitxer a descomprimir es un directori, per a efectuar la descompressió es fa el següent. es llegeixen dues líneas, on la primera és el nom original del fitxer comprimit i la segona és la mida del fitxer comprimit. seguidament es llegeixen els caràcters indicats per la mida i se li aplica la funció de descompressió de l'algoritme emprat. Es guarda el resultat (generant els directoris pares en cas de que no existeixin) i es repeteix fins al final del fitxer.

Diagrama de classes Capa de Domini

Aquest diagrama de classes té en compte totes les classes de la capa de domini. Fitxer és el controlador d'aquesta capa.



Capa de Dades.

En aquesta secció detalla les classes de la capa de dades. Aquesta capa només consta del controlador de la capa. En aquesta classe s'implementen totes aquelles funcionalitats relacionades amb l'acces al sistema de fitxers.

CtrlDades

Classe CtrlDades
Autor: Jordi Garcia Aguilar
Mètodes Públics:

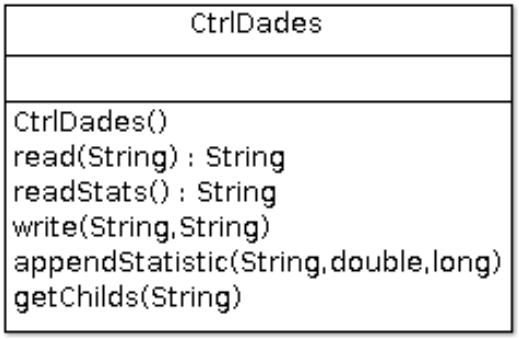
- **read():** Mètode per llegir un fitxer
  - **Pre:** La ruta absoluta al fitxer (String)
  - **Post:** Un String amb el contingut del fitxer
- **readStats():** Mètode per llegir el fitxer del històric d'estadístiques de compressions/descompressions.
  - **Pre: -**
  - **Post:** Un String amb el contingut del històric d'estadístiques.
- **write():** Mètode per escriure a un fitxer.
  - **Pre:** Contingut que es vol escriure (String), ruta absoluta al fitxer (String).
  - **Post** S'ha escrit al fitxer el contingut.
- **appendStatistics():** Mètode per escriure una entrada més al fitxer del històric d'estadístiques.
  - **Pre:** Ruta absoluta al fitxer d'entrada (String), identificador numèric de l'algoritme, grau de compressió/descompressió (double), duració de la compressió/descompressió en milisegons (long);
  - **Post:** El fitxer del històric d'estadístiques conté una linea més amb els valors de la nova entrada.
- **getChilds():** Mètode que recorre recursivament els fitxers d'un directori
  - **Pre:** Ruta absoluta al directori (String)
  - **Post:** Retorna un String amb tot els fitxers que conté el directori separats per el delimitador "//".

Mètodes Privats:

- **winOS():** Mètode per saber si el OS des d'on s'executa és Windows o no.
  - **Pre:** -
  - **Post:** *True* si es Os es Windows, *false* en cas contrari.

Diagrama de classes Capa de Dades

Aquest és el diagrama de classes de la capa de dades. Només hi ha el controlador de la capa CtrlDades.



Jocs de prova i Drivers

Tots els jocs de prova es troben a la carpeta **Jocs de proves** que es troba dins de la carpeta **EXE.**

Proves de compressió i descompressió.

Directori: Prova1
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la compressió d'un fitxer .txt de 5kb sense seleccionar un directori de sortida i amb qualsevol algoritme compatible amb un fitxer .txt.
Altres elements de la prova: Algoritmes
Fitxers de dades necessaris: 5kb.txt
Efectes estudiats. La compressió s'efectua correctament.

Directori: Prova2
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la compressió d'un fitxer .txt de 5kb, seleccionant el directori de sortida "output" i amb qualsevol algoritme compatible amb un fitxer .txt.
Altres elements de la prova: Algoritmes
Fitxers de dades necessaris: 5kb.txt, directori "output".
Efectes estudiats. La compressió s'efectua correctament.

Directori: Prova3
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la descompressió d'un fitxer .lz78, sense seleccionar ubicació de sortida.
Altres elements de la prova: Algoritme LZ78
Fitxers de dades necessaris: 5kb.lz78.

Efectes estudiats. La descompressió s'efectua correctament.

Directori: Prova4
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la descompressió d'un fitxer .lz78, seleccionant la ubicació de sortida "output".
Altres elements de la prova: Algoritme LZ78
Fitxers de dades necessaris: 5kb.lz78, directori sortida "output"
Efectes estudiats. La descompressió s'efectua correctament.

Directori: Prova5
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la descompressió d'un fitxer .lzw , sense seleccionar ubicació de sortida.
Altres elements de la prova: Algoritme LZW
Fitxers de dades necessaris: 5kb.lzw.
Efectes estudiats. La descompressió s'efectua correctament.

Directori: Prova6
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la descompressió d'un fitxer .lzw, seleccionant la ubicació de sortida "output".
Altres elements de la prova: Algoritme LZW
Fitxers de dades necessaris: 5kb.lzw, directori "output"
Efectes estudiats. La descompressió s'efectua correctament.

Directori: Prova7
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la compressio d'un directori "test" que conté fitxers i subdirectoris, amb qualsevol algorisme que no sigui el JPEG ja que no és compatible. Sense seleccionar ubicació de sortida.
Altres elements de la prova: Algoritmes
Fitxers de dades necessaris: directori "test".
Efectes estudiats. La compressió s'efectua correctament.

Directori: Prova8
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la compressio d'un directori "test" que conté fitxers i subdirectoris, amb qualsevol algorisme que no sigui el JPEG ja que no és compatible. Seleccionant ubicació de sortida "output"
Altres elements de la prova: Algoritmes
Fitxers de dades necessaris: directori "test", directori sortida"output".
Efectes estudiats. La compressió s'efectua correctament.

Directori: Prova9
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la descompressio d'un fitxer test.lz78 sense seleccionar ubicació de sortida.
Altres elements de la prova: Algoritme LZ78
Fitxers de dades necessaris: test.lz78.
Efectes estudiats. La descompressió s'efectua correctament.

Directori: Prova10
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la descompressio d'un fitxer test.lzw sense seleccionar ubicació de sortida.
Altres elements de la prova: Algoritme LZW
Fitxers de dades necessaris: test.lzw.
Efectes estudiats. La descompressió s'efectua correctament.

Directori: Prova11
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la compressio d'un fitxer "boxes.ppm" amb l'algorisme JPEG
Altres elements de la prova: Algoritme JPEG
Fitxers de dades necessaris: boxes.ppm.
Efectes estudiats. La compressió s'efectua correctament.

Directori: Prova12
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la descompressio d'un fitxer "boxes.jpeg".
Altres elements de la prova: Algoritme JPEG
Fitxers de dades necessaris: boxes.jpeg.
Efectes estudiats. La descompressió s'efectua correctament.

Drivers JPEG

Com demanava el full de la normativa, els drivers del JPEG es troben a la carpeta /FONTS/JPEGDrivers. S'afegeixen perquè són drives nous per les noves funcionalitats que s'han inclòs al JPEG i per a la nova classe Huffmann Tree.

Prova de Huffman.

Classe HuffmanTree
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la compressió i descompressió d'un vector amb Huffman.
Altres elements de la prova: HuffmanTree
Efectes estudiats. Es comprimeix i es descomprimeix prou bé.

Prova de JPEG. Compressió d'una fotografia blanca

Classe JPEGDriverBlanc
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la compressió i descompressió d'un fitxer blanc (tots els valors del color a 255).
Altres elements de la prova: Classes fitxer, HuffmanTree, Imatge
Fitxers de dades necessaris: Imatge PPM /EXE/Jocs de prova/ppms/blanc.ppm.
Efectes estudiats. La imatge es comprimeix i es descomprimeix prou bé

Prova de JPEG. Compressió d'una fotografia no blanca.

Classe JPEGDriverBlocs
Objecte de la prova: Aquesta prova vol comprovar com es duu a terme la compressió i descompressió d'un fitxer no blanc, una imatge normal.
Altres elements de la prova: Classes fitxer, HuffmanTree, Imatge
Fitxers de dades necessaris: Imatge PPM /EXE/Jocs de prova/ppms/boxes_1.ppm.
Efectes estudiats. La imatge es comprimeix però no es descomprimeix correctament. S'ha aconseguit trobar l'error.

Prova de JPEG. Compressió d'una fotografia no suportada.

Classe JPEGDriverP3
Objecte de la prova: Aquesta prova vol comprovar com la classe imatge bloqueja la compressió d'una imatge que no sap comprimir.
Altres elements de la prova: Classes fitxer, HuffmanTree, Imatge
Fitxers de dades necessaris: Imatge PPM /EXE/Jocs de prova/ppms/image.ppm.
Efectes estudiats. L'excepció salta correctament.