Hugo Freire Gil

E. T. en Informàtica de Sistemes

Consultor: Antoni Martínez Ballesté.

18 de Juny de 2004

Agraïments.

S'ha d'agrair a Internet, la possibilitat que ens dona de tenir accés a documents que d'altre manera serien molt difícils d'aconseguir, graciés a això s'ha pogut desenvolupar pràcticament tot el treball.

Resum.

L'objectiu principal del treball consisteix en aconseguir i documentar algun tipus de sistema que permeti incrustar en una imatge digital, un codi que la identifiqui de la resta de les copies, i que no suposi un canvi visual en la imatge. Per aquest propòsit, aprofitem la petita diferencia en la percepció que existeix entre tons de colors, per a realitzar petits canvis en las tonalitats dels punts que formen la imatge, i d'aquesta manera introduir codi ocult e imperceptible (marca d'aigua) per al ull humà. A més, aquest sistema ha de permetre llegir el codi, encara que es facin petites modificacions en la imatge.

Per a la segona part del treball s'intenta solucionar el problema que pot causar que dos propietaris d'una mateixa imatge, però amb diferent marca, puguin crear una imatge com a producte de mesclar les dues, i que el resultat no pugui ser identificat per el nostre programari. En aquest punt s'estudiaran dos tipus de programari, un que ens servirà per a simular una confabulació entre dos propietaris, i unaltre que ens permetrà identificar almenys un dels dos confabuladors.

Índex

1. Introducció

1.1. Justificació del treball de final de carrera i context en el qual es desenvolupa.

En la era actual en la que ens trobem, la transmissió d'informació entre punts distants no suposa un gran problema, la proliferació de les xarxes publiques, l'abaratiment dels productes informàtics i la introducció, cada vegada més d'hora, de la gent en aquest món digital, han posat a l'abast de tothom tecnologies molt sofisticades. Tecnologies capaces de replicar tot tipus de material que abans només estaven a l'abast de governs i institucions militars. Molta gent, animada per l'anonimat que proporciona Internet, i la proliferació de la creença de que no cal pagar res que es pugui aconseguir gratuïtament, tot hi que això signifiqui atentar contra lleis de propietat intel·lectual, han fet davallar un dels pilars sobre el que es sustenta la nostre societat, la inversió en creativitat. Però com es pot protegir informació no tangible?, informació quantificada i concreta, representada per simples canvis d'estats, dipositat en contenidors que per el simple fet de mostrar la informació que contenen, revelen pas a pas tot el camí necessari per a duplicar-lo i transmetre-ho. Doncs la solució encara no existeix, en la història digital ens trobem exemples de molts intents fracassats, que poc a poc han vist com la tecnologia trobava la manera de saltar qualsevol protecció. Per aquest motiu, s'han orientat els esforços cap altres objectius, desenvolupar tècniques que no protegeixen la informació contra la duplicitat però si mantenen la seva autoria i poder demostrar qui va ser el seu propietari original i d'aquesta manera poder castigar els que impunement es dediquen a treure profit de la falsificació digital de qualsevol treball creatiu.

Primer es van desenvolupar sistemes molt sofisticats i molt visibles, que feien molt difícil la falsificació i que els identificava molt fàcilment, com els que fan servir els bitllets, i cada vegada més, es fa tot lo contrari, crear marques identificatives molt senzilles però molt difícils d'observar, marques que només amb els utensilis adequats poden ser descobertes, aprofitant-se de la creença de que si no la podem observar no existeix.

En aquest treball el que s'intenta és crear un sistema de protecció de la propietat intel·lectual en imatges digitals, mitjançant el que es coneix amb el nom de "marca d'aigua", provar la seva fortalesa contra alguns tipus d'atacs, i trobar una possible solució contra alguns d'ells. El treball no pretén obtenir un mètode infalible però si podria ser un bon començament per a desenvolupar-ne un de molt robust.

1.2. Objectius del treball de final de carrera.

Entre els objectius que s'haurien d'assolir en aquest treball cal destacar els següents:

- Conèixer les principals tècniques de marca d'aigua per a la protecció del copyright en imatges digitals.

- Aprofundir en mètodes per evitar la eliminació 'aquest tipus marques d'aigua, tant per atacs involuntaris contra el sistema, per alteració d'una imatge, com per atacs premeditats de dos confabuladors.

- Implementar el programari necessari per poder introduir una marca d'aigua sobre una imatge digital i poder recuperar, posteriorment, el identificador de la copia.

- Implementar el programari necessari per a poder simular un atac contra el sistema mitjançant dos confabuladors.

- Implementar el programari necessari per a contrarestar l'atac per confabulació.

- Descobrir les febleses d'aquest sistema de copyright i intentar de millorar els resultats.

1.3. Enfocament i mètode seguit.

Se li ha donat al projecte un enfocament força realista, no s'ha intentat buscar una solució innovadora, sinó que s'ha tractat com una excusa per a conèixer en general l'estat actual en que es troba el àmbit de les marques d'aigua.

Per aconseguir els objectius establerts, s'ha abordat cadascun com un de sol i independent dels altres. En aquest sentit, s'ha delimitat cadascun dels problemes i s'han fixat dates límits per aconseguir solucionar-los. En tots els punt s'ha començat per una cerca exhaustiva d'informació sobre les actuals solucions a aquest problemes i s'ha optat per la que millor s'ajustés als límits de temps i de requisits, evitant d'aquesta manera que alguns problemes esdevinguessin massa extensos en temps i recursos. Això ha permès d'obtenir la millor solució amb el menor esforç.

Una vegada trobat el camí adequat per a cada objectiu s'ha començat amb el disseny i l'adaptació de la solució obtinguda per tal de apropar-la a les nostres necessitats, obtenint en molts casos millores sobre la idea original. Havent obtingut el disseny s'ha començat amb l'anàlisi i la programació. El fet d'haver escollit Java com ha llenguatge de programació per a codificar el projecte ha estat molt encertat, ja que hi ha una alta disponibilitat de llibreries de lliure accés, que han permès assolir molts objectius abans del temps establert. A més, aquest fet ens han servit per disposar d'un producte independent de la plataforma escollida.

En acabar la programació s'ha començat a fer les probes pertinents i s'han fet les modificacions necessàries en el codi per solucionar els problemes obtinguts. Una vegada ja comprovat el correcte funcionament del mòdul s'ha continuat amb el següent, fins acabar totes les parts que componen el projecte.

1.4 Planificació del projecte.

En un principi es va pensar en una planificació diferent començant per fer l'estudi de cadascun dels objectius, després el disseny de tots, l'anàlisi i la programació, però finalment es va decidir que la millor manera era dividir el problema i tractar cada divisió independentment i no continuar amb la següent fins tenir-la completament preparada.

Per aquest motiu es va establir que els cinc primers dies es dedicarien a la cerca d'informació sobre els fitxers bmp i els vuit següents per al disseny d'unes petites funcions per a llegir i modificar aquest tipus de fitxer. Per al 23 de març s'havia de tenir el primer mòdul.

En la segona part es va pensar que es necessitarien vuit dies més per trobar un sistema de marca d'aigua adient, i quinze dies per a dissenyar-lo. Cap al 15 d'abril s'hauria de començar la codificació que finalitzaria sis dies després. Un parell de dies més per fer les proves adients al sistema de marca d'aigua.

Pel següent punt, el termini era de divuit dies, necessaris per entendre els codis d'Hamming i més concretament els duals. Pel disseny de la part que servirà per a detectar confabuladors se li va donar un marge de deu dies, i per a la seva implementació cinc dies més. Cap al dia 30 d'abril ja hauria d'estar provada aquesta part.

Per últim quedava el programa que s'encarregaria de crear les imatges dels confabuladors. Per el seu disseny es disposava de cinc dies, i per a la programació quatre dies més. Les proves haurien de durar fins al 16 de març. Els últims dies es dedicarien a la realització d'aquesta memòria i la presentació final.

1.5 Productes obtinguts.

Del desenvolupament d'aquest treball han resultat dos productes. El primer d'ells és un programa en Java, capaç d'encastar una marca d'aigua en una imatge, mitjançant un sistema estadístic. Aquest mateix programa, disposa d'una funció per crear codis duals per a la marca d'aigua, que queden enregistrats en un fitxer que relaciona codi amb client. Un altre mòdul d'aquest programa ens permet llegir la marca encastada en la imatge i descobrir un dels propietaris encara que s'hagi fet servir algun tipus de confabulació entre dos usuaris que disposen de una mateixa imatge però amb diferent codi d'identificació. Aquest sistema de lectura i escriptura de la marca d'aigua no necessita la imatge original per a obtenir el codi ocult, això facilita la seguretat del sistema al no haver de presentar la imatge original per demostrar el seu origen.

El segon producte, és també un programa en Java dissenyat per a crear una confabulació entre dos usuaris, per tal d'ocultar el propietari. Aquest simplement obté una imatge per mitjà de mesclar dues imatges iguals amb marques d'aigua diferent.

1.6. Breu descripció dels capítols de la memòria.

En els capítols següents es fa una descripció detallada de cadascuna de les parts que conté el projecte, començant pel la manipulació d'imatges digitals fins a l'ús de codis duals d'Hamming.

Cada punt principal estarà dividit en ivisions en les que es descriurà en primer lloc l'estat actual de l'art en el tema treballat, on es farà una petita volta per alguns dels sistemes utilitzats en aquest àmbit. La següent divisió detallarà el camí escollit, el perquè, i les seves avantatges sobre altres. Seguidament és farà una relació del punt amb el codi obtingut i es comentaran algunes parts rellevants i la seva relació amb altres mòduls del projecte. Altre punt important són els objectius assolits, es a dir, el que s'ha aconseguit i el que no ha estat possible o necessari. A l'apartat de problemes es comentaran quines dificultats s'han trobat en el disseny i quins problemes podrien aparèixer amb el seu us. I Finalment es mencionen les millores que es podrien fer per pal·liar aquests problemes i quin podria ser el seu futur.

Un últim capítol general, recollirà les conclusions a les que s'ha arribat, seguit dels annexos amb el codi font comentat.

2. Lectura i modificació d'imatges digitals.

Actualment sembla impensable per a qualsevol fotògraf, no fer us de les noves eines digitals de retoc fotogràfic, que permeten al professional aconseguir efectes impensables fins fa poc o molt difícils d'aconseguir amb els sistemes tradicionals. Aquest tractament digital fa imprescindible l'emmagatzematge d'aquestes imatges en suports electrònics, per això es van desenvolupar molts estàndards, per tal de minimitzar els costos que provocaven.

2.1. Tipus de fitxer per emmagatzemar imatges digitals.

En la actualitat es disposa de centenars de sistemes d'emmagatzematge d'imatges digitals, des dels més senzills com els fitxers de mapa de bits, en els que podem trobar els BMP (bitmap) o els TIFF (Tag Image File Format), que basen la seva filosofia en guardar el màxim possible de la imatge digital en detriment de l'espai a ocupar, fins els fitxers més complexos com els JPEG (Joint Photographic Experts Group) que intenten estalviar espai fent servir una propietat de totes les imatges, que molts dels punts que les formen tenen la mateixa tonalitat de color. Tot i el estalvi en espai, en el mon digital es continuen fent servir els primers formats degut a la gran qualitat d'imatge que proporcionen, encara que , cada vegada més, sembla que la tendència va cap a l'estalvi d'espai mitjançant els sistemes comprimits, degut principalment, a la necessitat de transmetre-les a través de la xarxa d'Internet. Aquest es precisament, el motiu d'aquest treball, intentar controlar la circulació d'aquestes imatges.

2.2. Camí escollit.

Degut a la complexitat afegida que té la manipulació d'imatges comprimides i la exactitud que proporcionen els mapa de bits, s'ha decidit encarar el treball cap la protecció d'imatges en format BMP, encara que les llibreries utilitzades per a la manipulació de les imatges permeten treballar amb fitxers JPEG, GIF, TIFF, etc... i que el sistema creat podria funcionar també en aquests formats per a fotografies amb molt bona qualitat d'imatge.

2.3. Mòduls del programari relacionats.

En aquest mòdul del programari el que s'intenta es crear un objecte *java.awt.image.WritableRaster*, que ens permetrà manipular la imatge d'entrada original. Per aquest objectiu s'ha creat una classe anomenada *utilimatge*, amb la

que podrem carregar una imatge des d'un fitxer i guardar-la en un altre després de fer les modificacions adients. Aquesta classe treballa amb *java.awt.image.RenderedImage*, que tindrem que transformar en un *java.awt.image.WritableRaster* a la classe principal del programa anomenada *imatgeCopiright*. En aquesta part del projecte també es van preparar les primeres classes per a la interacció amb l'usuari com ara la classe *GUI* y *errors*, la primera serveix per a mostrar finestres per entrar informació i treure missatges, i la segona per a controlar els diferents errors que podrien ocórrer, com fitxers que no existeixen, encara que aquesta classe, ara per ara, només treu directament els missatges d'error per la sortida estàndard.

2.4. Objectius aconseguits.

L'objectiu principal d'aquesta part era poder llegir una imatge BMP, preparar-la per a poder-la modificar i finalment guardar-la en unaltre fitxer. Per tant, aquest objectiu l'hem acomplert completament, fins i tot, gracies al *awt* de Java es podrien llegir pràcticament qualsevol fitxer d'imatge com ara els JPEG, TIFF, GIF, etc...

2.5. Problemes.

El principal problema en aquest mòdul va ser aconseguir el nivell necessari per a poder treballar amb la llibreria *awt* de Java, ja que la imatge original carregada ha de passar per alguns processos abans de poder ser manipulada. Pràcticament el 90 per cent del treball en aquesta secció es va dedicar a llegir la documentació del *awt* i fer les probes necessàries amb aquesta llibreria.

2.6. Possibles millores.

Tot i que el programa permet llegir molts dels tipus més famosos de fitxers d'imatges digitals, no passa el mateix a l'hora de guardar-lo, en aquest sentit el programa només permet guardar la imatge en un fitxer BMP, encara que l'únic que s'hauria de fer es preparar-lo per demanar al usuari el tipus de fitxer en el que vol guardar la imatge (o detectar-lo per la extensió del fitxer a guardar) i canviar la variable *format*, que es troba a la classe *imatgeCopyright* a la línia 82, per el nom de format adient ("JPEG", "GIF", "TIFF").

3. Introducció i lectura d'una marca d'aigua en una imatge digital.

L'alt desenvolupament que han sofert les comunicacions en els últims vint anys, l'abaratiment dels preus i més concretament, l'arribada d'Internet a qualsevol lloc, han aconseguit una mena de globalitat que permet la lliure circulació de qualsevol tipus d'informació a través de la Xarxa. Informació que pot tenir drets de copia, com el cas que ens ocupa, les imatges digitals. Evitar aquesta circulació sembla impossible, però amb un sistema de signatura que identifiqui una imatge (marca d'aigua), podríem obligar, a les persones a les que les hem venut un dret sobre una imatge, a no difondre aquestes imatges sense precaució, ja que els hi podríem demanar responsabilitats sobre aquests fets.

3.1. Situació actual.

En l'actualitat hi ha molts sistemes de marca d'aigua, fins i tot per a altres medis com el vídeo o el àudio. En el cas que ens ocupa, la imatge digital, en podem trobar un món. El sistema de protecció del copyright, conegut com a "marca d'aigua" o *Watermark,* consisteix en introduir una modificació en la imatge original per tal d'identificar al propietari del mateix. Es podrien classificar en dos grups principals, sistemes de marques d'aigua visibles i sistemes de marques d'aigua invisibles. Els primers consisteixen en introduir una modificació molt visible a la imatge original, de tal manera que esborrar la marca seria impossible i produiria una pèrdua molt important en la qualitat de la imatge. Els mètodes que corresponen al segon grup, consisteixen en un sistema que permet introduir petites modificacions en la imatge original de tal manera que siguin impossibles de detectar a simple vista. Aquest mètode ens permet mantenir l'aspecte de la imatge, molt més adient per als clients.

Marca d'aigua visible.
"The USC-SIPI Image Database"


Marca d'aigua invisible.
"The USC-SIPI Image Database"

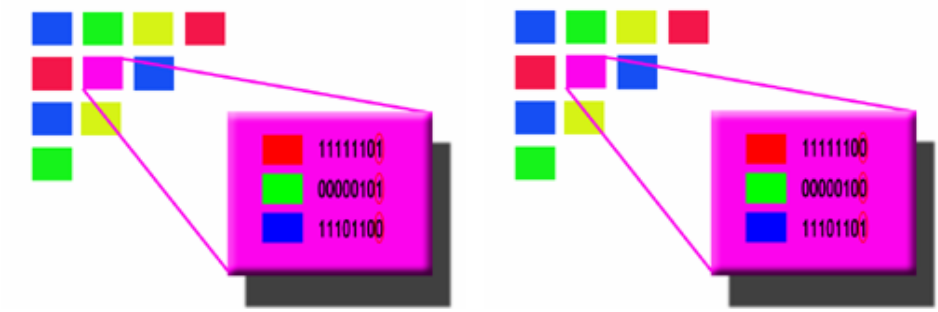
Entre els mètodes de marques d'aigua invisibles podem trobar dos tipus més, els que necessiten la imatge original per treure la marca i els que no. El

segon grup te un avantatge addicional sobre el primer ja que no s'ha de mostrar la imatge original per a demostrar que n'és el propietari, per exemple en un judici, i la pot guardar a una caixa forta a la que ningú no tindrà accés mai. Entre els primers trobem sistemes com el LSB (Least Significant Bit) que consisteix en canviar l'últim bit del color de cada píxel, segons si el que volem introduir és un 1 o un 0. Aquesta modificació no suposa un canvi visible en la imatge i ens permet mitjançant la comparació amb la original detectar quin es el missatge ocult. Aquest sistema te el problema de ser molt sensible a les modificacions de la imatge i normalment es repeteix la marca moltes vegades a la imatge per tal de solucionar-lo. En el segon grup podem trobar sistemes estadístics i sistemes d'espectre dispers (Spread-Spectrum), aquest últim sembla ser el més robust de tots i més difícil de detectar, les actuals investigacions giren cap aquesta banda.

A més d'intentar protegir els drets d'autor també s'intenta protegir els drets dels clients contra un mal ús d'aquest sistema, ja que alguns autors podria introduir una marca d'aigua d'alguns dels seus clients en una imatge i distribuir-la per tal de cobrar més endavant possibles indemnitzacions per un mal ús. En aquest sentit s'han empleat mètodes que permeten incrustar varies marques d'aigua i sistemes d'encriptació de claus publiques.

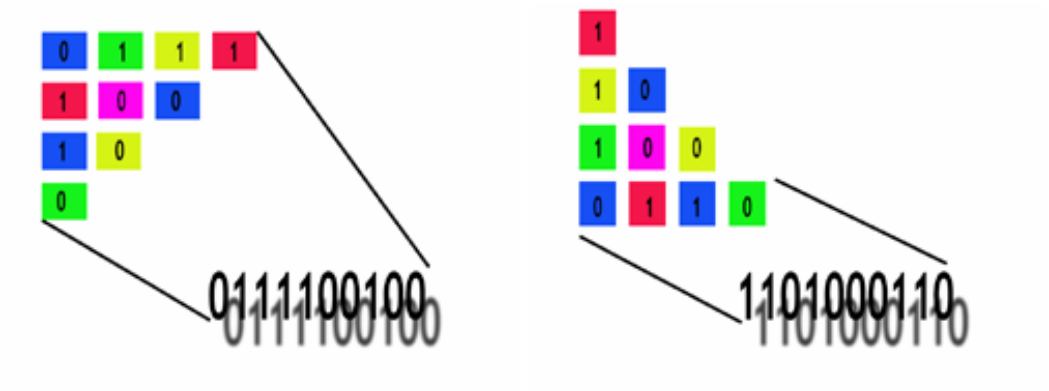
3.2. Direcció del treball.

En un principi va semblar que el mètode més senzill d'implementar era el LSB, però en començar a fer el disseny van començar a aparèixer els primers problemes, principalment en la detecció de la marca d'aigua. Aquest sistema consisteix en modificar l'últim bit de cada un dels colors que formen un píxel segons si volem introduir un 1 o un 0.



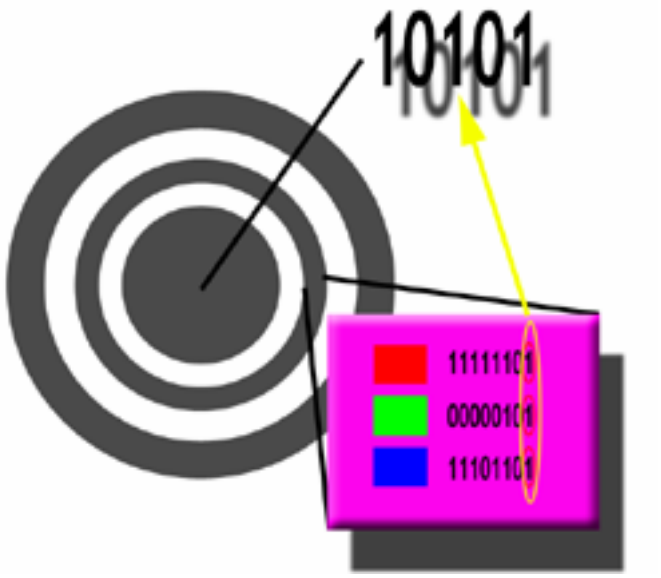
En aquest exemple volem introduir un 1 per tant canviem els últims bits dels colors del píxel (per introduir redundància)

Fins aquí cap problema, la dificultat es troba quan volem llegir la seqüència de bits amagats en la imatge, ens trobem que si la imatge ha estat retallada o girada no som capaços de comparar-la amb la original, fins i tot, si aconseguim un sistema que no necessiti aquesta imatge, se'ns fa molt difícil de llegir la seqüència en ordre.



No podem llegir els bits en el ordre adequat quan girem la imatge

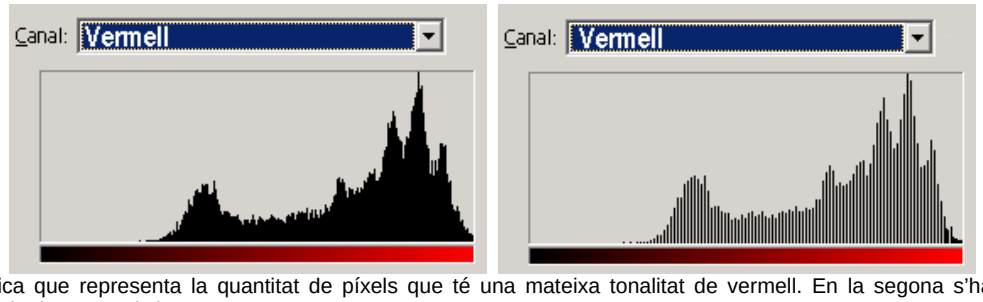
Arribat a aquest punt s'havia de buscar unaltre solució. Per evitar el problema al girar la imatge s'ha pensat en una mena de marca d'aigua circular que consisteix en no escriure en horitzontal sinó en forma radial, partint d'un centre i anant canviant els píxels de manera que si l'últim bit dels colors d'un píxel són zeros el bit es 0 i si són uns el bit es un 1.



Amb aquest sistema evitem utilitzar la imatge original i també evitem el problema que sorgeix quan girem la imatge

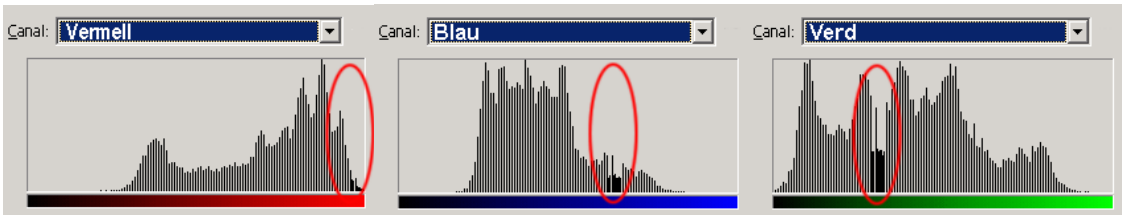
Encara així continuem tenint un problema: com localitzem l'origen de la marca? Per aquest motiu s'ha deixat, també, aquest sistema de banda, i s'ha cercat un de nou.

Finalment, s'ha optat per un sistema estadístic, que consisteix en comptar la quantitat d'un mateix color que hi ha en la imatge, si la quantitat d'un color no supera un mínim tindrem un 0 en cas contrari tindrem un 1. Com que no podem fer grans modificacions en els píxels de la imatge ens limitarem a canviar el color d'un píxel incrementant-lo en una tonalitat, només si aquest conte un color que correspon a un nombre parell i volem introduir un 0. D'aquesta manera només aniran quedant colors parells amb píxels si el bit que correspon a aquest color és un 1.



Gràfica que representa la quantitat de píxels que té una mateixa tonalitat de vermell. En la segona s'ha aplicat el mètode de marca d'aigua.

Amb aquesta solució la obtenció de la marca d'aigua introduïda no depèn de la orientació de la imatge, ni de la mida respecte l'original, ni tan sols cal que la imatge sigui completa (fins a una porció d'un 10 % de l'original). L'únic problema el trobem amb imatges molt obscures o molt clares, que no contenen tot l'espectre de tonalitats. Per a solucionar-lo s'han empleat les altres bandes que contenen el color del píxel amb un desplaçament inversament proporcional al nombre de bandes per píxel.



Es pot observar com en la banda dels vermells quasi no es diferencien els bits però observant la banda dels verds es veu perfectament

3.3. Mòduls del programari relacionats.

En aquesta part del treball es van desenvolupar les classes *grabarmarca* i *llegirmarca*, que serveixen per gravar una sèrie de bits en la imatge i per a llegir-los, respectivament. La classe *grabarmarca* necessita una imatge en forma de *writableRaster* i una seqüència de bits agrupats per bytes. S'ha de cridar la funció *marcar* d'aquesta classe amb els arguments anteriors per tal de marcar la imatge. Aquesta funció s'encarrega de llegir tots els píxels i comprovar els colors de totes les bandes, si es tracta de tonalitats parelles (últim bit igual a 0) es llegeix el bit que correspon en la marca que es vol introduir i es canvia l'últim bit del color si el bit de la marca es 0. A aquesta classe també podem trobar una funció, *extreurebits*, que s'encarrega de treure tots els bits dels bytes que enviem a *marcar*.

Per llegir la marca d'una imatge s'ha creat *llegirmarca,* amb la seva funció *llegir*, que haurà de rebre un *writableRaster*, i retornarà una seqüència de bits agrupats per bytes. El primer que fa, es guardar totes les dades corresponents a cadascun dels píxels en una matriu que contindrà la quantitat de píxels d'un mateix color. Després llegim les quantitats de les tonalitats múltiples de dos,

sumant-li a cada una les que li corresponen dels altres colors (verd i blau, en el cas de les imatges en color RGB).Seguidament, fem lo mateix amb la següent tonalitat, i fem una divisió entre el primer càlcul obtingut i aquest. Si el resultat supera un coeficient, tindrem un 1 i sinó un 0. D'aquesta manera aconseguirem més flexibilitat per el cas en que les imatges s'hagin modificat després incrustar la marca.

El programa es capaç de llegir i guardar la marca d'aigua en qualsevol tipus d'imatge i amb qualsevol quantitat de bandes, fins i tot en tonalitats de gris.

3.4. Objectius aconseguits.

Un dels objectius d'aquest mòdul era aconseguir gravar una seqüència de bits en una imatge sense que aquesta tingués cap canvi visual, per aconseguir-ho s'ha empleat un sistema estadístic. L'altre objectiu consistia en llegir aquesta seqüència de bits. En tots dos casos els objectius s'han acomplert exactament, ja que en totes les probes realitzades s'han obtingut resultats positius. El sistema, a més, és capaç d'introduir a la imatge fins a 128 bits d'informació.

3.5. Problemes.

Tot el problema d'aquesta fase es trobava en aconseguir un mètode de marcatge adient, amb la suficient robustesa com per a sobreviure a la majoria dels canvis que podria sofrir una imatge. En aquest punt, la recuperació de la marca d'aigua resultava crítica, i la majoria dels primers sistemes no permetien recuperar-la adequadament. Finalment, en trobar el mètode correcte, no va haver gaire més problemes en la fase de codificació.

Pel que fa al mètode que es fa servir per encastar la marca d'aigua s'han trobat problemes amb programes de retoc fotogràfic, degut a que aquests tendeixen a suavitzar les colors quan es giren les imatges o es redueixen, però no succeeix amb programes que només es limiten a girar la imatge i reduir-la sense preocupar-se per el resultat final.

3.6. Possibles millores.

Degut a que els programes de manipulació d'imatges digitals intenten suavitzar l'aspecte final d'una imatge, ens trobem que en retocar una imatge que conte una marca d'aigua, aquesta es perd ja que el programa intenta acostar els colors dels píxels de la imatge. Per a poder superar aquest

problema, s'hauria de aconseguir que no es limités només a canviar les tonalitats parelles sinó que hauria de fer que les tonalitats contigües dels píxels tendeixin cap una tonalitat de color, per tal d'evitar esglaons molt pronunciats que aquests programes reduirien. A l'hora de llegir la marca, hauríem de llegir un conjunt de tonalitats, i mirar si aquestes tendeixen cap a un extrem o cap a unaltre del grup i determinar d'aquesta manera si es tracta d'un 0 o un 1.

Aquest son alguns exemples de les proves realitzades amb el programa.

Aquí podem veure la imatge original, una imatge amb la marca d'aigua i unaltre imatge resultant d'agafar una porció de la segona imatge i voltar-la. (The USC-SIPI Image Database)

Imatge original en blanc i negre, seguida d'una amb una marca d'aigua i una altre producte de reduir la segona i voltar-la (The USC-SIPI Image Database)

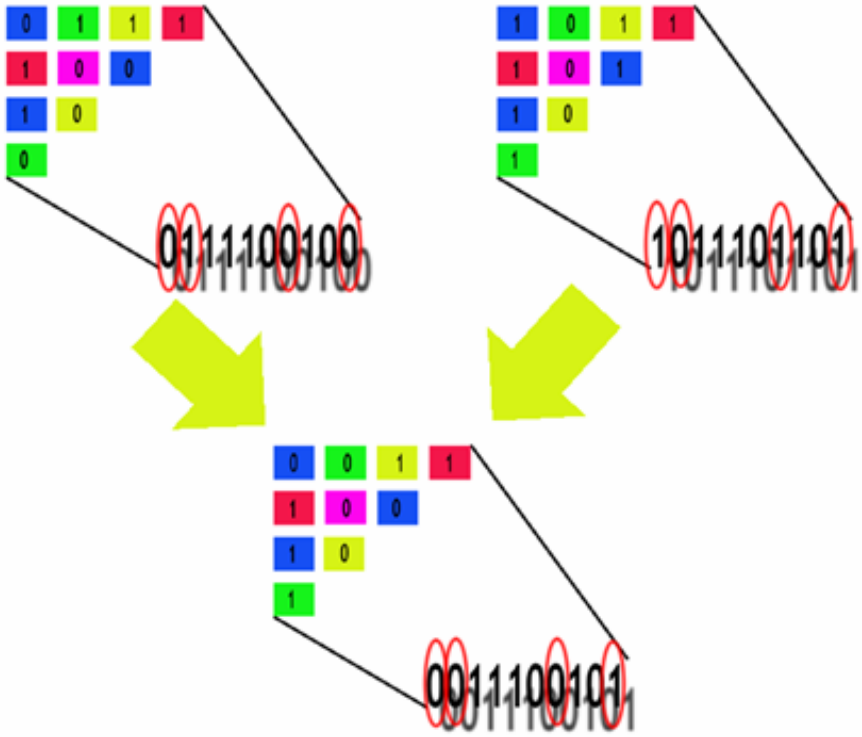
En aquestes proves tots els resultats han estat positius i s'ha pogut obtenir la marca d'aigua completa.

4. Sistema de confabulació.

Un dels principals problemes que tenen les marques d'aigua es la possibilitat de que dos individus que tenen una mateixa imatge però amb una marca d'aigua diferent, s'associïn per tal de mesclar les seves imatges i obtenir una tercera que no conté cap de les marques originals.

4.1. Camí escollit.

El sistema de confabulació es molt senzill, només cal que comparem dues imatges i creem unaltre amb píxels escollits aleatòriament d'una o d'altre imatge.



El codi obtingut de la confabulació no podria identificar als confabuladors

El resultat no podria identificar al propietari de cap de las dues imatges.

4.2. Mòduls del programari relacionats.

Per acomplir aquesta tasca s'ha pensat en una petita classe que carregarà dos imatges iguals però amb diferent marca d'aigua, i comparará píxel a píxel, en el moment que un d'aquests píxels difereixi, el programa farà servir una funció aleatòria que escollirà un dels dos. La classe utilitzada es *Confabuladors*, i es tracta d'una classe que es pot executar independent.

5. Codis de Hamming duals per a contrarestar la confabulació.

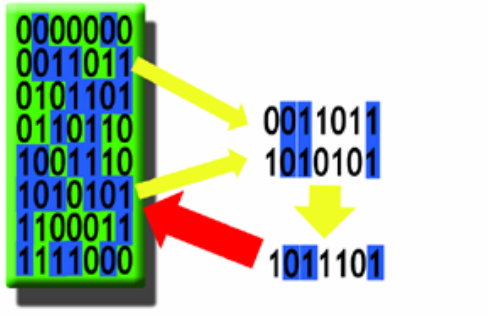
La robustesa d'un sistema de marcatge pot veure's compromès per l'atac conjunt de dos usuaris, per aquest motiu s'ha de desenvolupar un mètode que ho contrarestí. En aquest sentit sembla que la millor solució es crear uns codis que identifiquin als usuaris encara que es modifiquin algunes parts d'aquest codi.

5.1. Els codis d'Hamming.

Per a poder controlar aquest tipus de confabulació actualment s'utilitzen codis d'Hamming, són els mateixos codis que s'utilitzen per a la transmissió de dades i que permeten descobrir de quin codi original es tracta encara que aquest es modifiqui en passar a través del medi de comunicació. Aquest codis es basen en la propietat de que qualsevol paraula que pertany al codi, te una diferencia mínima de bits amb qualsevol altre paraula del codi, aquesta propietat es coneix com a distancia mínima d'Hamming i permet reconèixer el codi original encara que aquest hagi sofert algunes modificacions. Els més utilitzats són els codis duals un subconjunt d'aquests codis que tenen la particularitat que la distancia de Hamming mínima i màxima és la mateixa.

5.2. Els codis del programari.

La solució al problema dels confabuladors passa per forçar a que els bits introduïts mitjançant la marca d'aigua, acompleixin la propietat dels codis duals. Així, encara que dos confabuladors intentin crear una imatge mitjançant la mescla de dues, podem assegurar que la diferencia que trobaran entre una i l'altre, seran els bits que corresponen a la distancia mínima de Hamming i els altres restants no els podran detectar. Aquests bits que no han detectat els confabuladors ens serviran per a identificar a un dels dos propietaris, encara que seria possible trobar els dos.



El codi detecta a un dels dos codis originals ja que te més probabilitats per que tres dels bits li corresponen per defecte.

5.3. Mòduls del programari relacionats.

En aquesta fase del treball s'han desenvolupat dues parts, una relacionada amb la codificació dels codis d'Hamming duals i altre per a l'emmagatzemament de les dades dels clients i els codis identificadors.

Per a la primera part s'ha creat una nova classe anomenada *codidual* que l'utilitzem per a transformar un enter en el seu corresponent codi dual. Per a dur-lo a terme, aquesta classe llegeix un fitxer anomenat *matrgen.icx* que conté una matriu generadora que utilitza per a crear aquests codis. En aquest fitxer s'indiquen les dimensions de la matriu i la matriu mateixa. D'aquesta manera si volem crear codis duals per a una llista més gran d'usuaris l'únic que hem de fer es introduir una matriu generadora de les dimensions adequades. Amb la funció *codificar,* aquesta classe multiplica el enter introduït per la matriu i retorna els bits (codi dual) que identificarà al client.

En la mateixa classe s'inclou una funció *mouBitsEsquerra* per a fer moviments de bits cap a l'esquerra en una llista de bytes. Ens serveix per fer operacions amb la matriu i el codi identificador.

La segona part d'aquesta fase es dedica a crear una petita base de dades per a introduir els codis duals de cada client i un nom que els identifica. Per aquest propòsit s'han creat les classes *IOFitxer* i *bdclients*. La primera classe serveix per obrir fitxers de lectura, fitxers d'escriptura, introduir i llegir enters, caràcters i bytes. La segona classe ens serveix per moure'ns pel fitxer de base de dades, per introduir clients, per llegir les dades dels clients (nom, codi identificador), canviar de registre, etc... Aquesta ultima classe utilitzar *IOFitxer* per el seu propòsit.

5.4. Objectius aconseguits.

L'objectiu d'aquest mòdul era crear un sistema que identifiques el propietari d'una imatge encara que s'hagi fet servir el programa de confabulació, en aquest sentit l'objectiu s'ha acomplert encara que els sistema no és completament fiable per que hi ha probabilitats de que aleatòriament el codi obtingut per els confabuladors correspongui a un altre client.

5.5. Problemes.

Els principals problemes que es van trobar a l'hora de desenvolupar aquesta part del treball van estar relacionats amb l'obtenció d'un codi

Hamming dual per a l'aplicació, aquesta fase passava primer per entendre el fonaments d'aquests codis i després per la obtenció d'una funció que els generés. La solució final va ser desenvolupar l'aplicació de forma que els codis obtinguts fossin producte d'una matriu generadora.

5.6. Possibles millores.

L'aplicació està preparada per a generar qualsevol tipus de codi dual només canviant la matriu generadora i canviant la variable MAXCLIENTS que depèn d'aquesta matriu. La matriu actual només permet un màxim de 8 clients, per tant la primera millora seria crear una matriu més gran que pogués permetre fins als 65536 clients i amb suficients bits redundants com per aconseguir uns resultats molt més satisfactoris.

També es podrien fer millores en la banda de la base de dades, afegint altres dades personals dels clients (direcció, telèfon, etc...).

6. Conclusions.

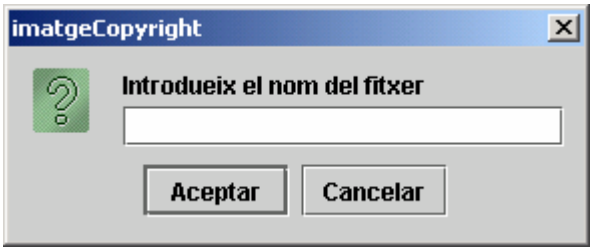
En la era actual en la que vivim la informació es un valor molt preuat, cada vegada més, els actes delictius s'aproximen cap aquest àmbit. La proliferació d'Internet, ha fet possible que aquesta informació viatgi ràpidament a qualsevol lloc del món, i de vegades aquesta te uns drets que no es respecten. Per a contrarestar-lo la comunitat científica ha continuat desenvolupant sistemes de protecció de dades des de fa molts anys, però ara per ara cap d'ells s'ha sortit victoriós. Sembla que una bona opció és intentar coaccionar als usuaris per evitar que transmetin sense problemes la informació confidencial de la que disposen. En aquest sentit la firma digital de documents, obliga als usuaris a anar amb més compte per evitar tenir responsabilitats. En el cas de la firma digital d'imatges es tan important el poder demostrar qui es el creador com qui ha estat responsable de difondre-les sense permís. Per aquest motiu els sistemes de marques d'aigua s'han estat desenvolupant en els últims anys, tant per imatges digitals, com vídeo i àudio. Els principals problemes que troben aquets sistemes són la modificació involuntària de l'original i l'atac directa contra el sistema de marca d'aigua. Per a solucionar el primer problema s'han desenvolupat mètodes cada vegada més robustos que suporten qualsevol tipus de modificació que mantingui l'essència de l'original, que a més els fa indetectables per els profanadors. L'atac directa contra la marca d'aigua pot ser contrarestada amb un bon sistema de marca però no es fiable quan el usuari disposa de dos originals amb diferents marques.. Per a solucionar-lo sembla que la utilització dels codis d'Hamming són una bona opció ja que ens permet obtenir bons resultats per a la majoria d'aquests atacs per confabulació, encara que no es del tot fiable.

Alguns punt importants que es podrien desenvolupar podria ser un tipus de programari que navegués per Internet buscant copies de l'original i llegint les marques d'aigua per informar de qui fa servir les copies i si hi te dret per a utilitzar-les. També seria interessant crear una marca d'aigua capaç de suportar la seva impressió en paper, o l'escaneja't, d'aquesta manera es podrien evitar altres tipus de frau.

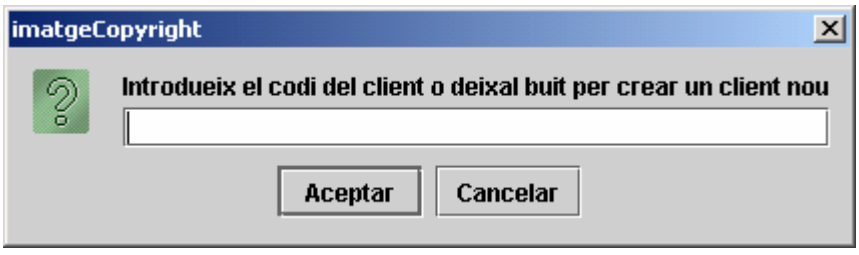
7. Manual d'utilització del programa

Fer servir el programa es molt senzill. Per executar la part d'incrustació de la marca d'aigua i la lectura només és necessita utilitzar la instrucció *java imatgeCopiright* i obtindrem al primera pantalla.

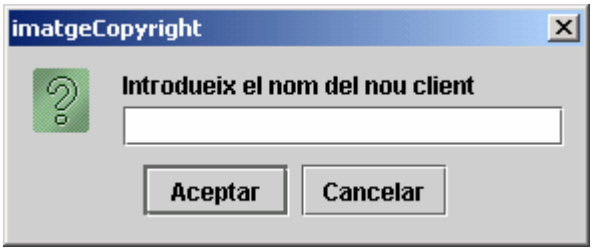
Aquí tenim dues opcions: introduir una marca d'aigua en un imatge o llegir la marca d'aigua. Polsant el 1 i retorn passarem a introduir la marca d'aigua i obtindrem una pantalla que ens demanarà la imatge original, aquí hem d'introduir el nom complet i la seva ruta fins el fitxer si cal.



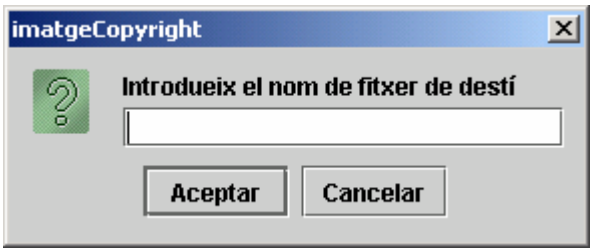
Una vegada em introduït el nom del fitxer original, l'aplicació ens demana el codi de client (del 1 al 8) o podem deixar-lo en blanc per crear un de nou amb un codi nou.



En el cas que creem un de nou se'ns demanarà el seu nom.

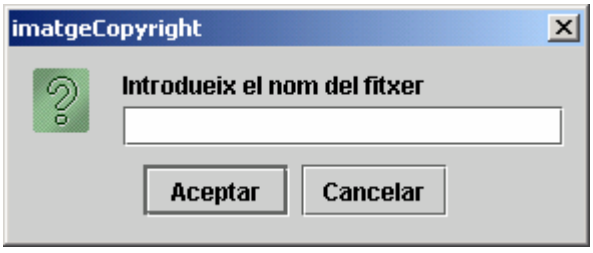


En aquest procés s'ha creat el codi dual que identificarà al client i s'ha guardat en la base de dades. Finalment ens demana el fitxer de destí on volem guardar la nova imatge amb la marca d'aigua incrustada.



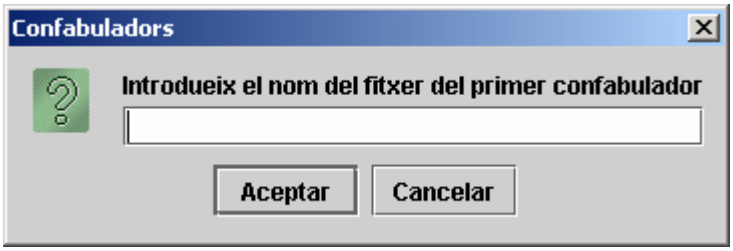
Si tot anat correctament, en aquest moment s'ha creat un nou fitxer amb la nova marca d'aigua.

Si el que volem es llegir la marca d'aigua, només hem d'utilitzar la opció 2 del menú principal. En aquest moment ens demana el nom del fitxer que conté la marca d'aigua.

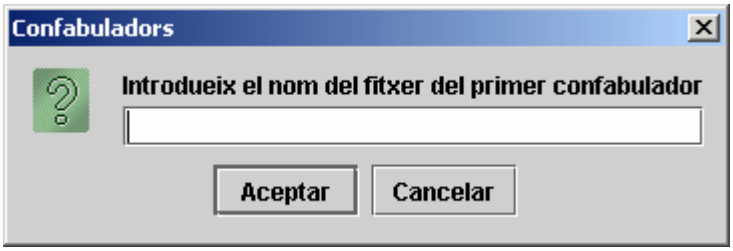


L'aplicació retornarà el nom del client al que pertany la marca d'aigua.

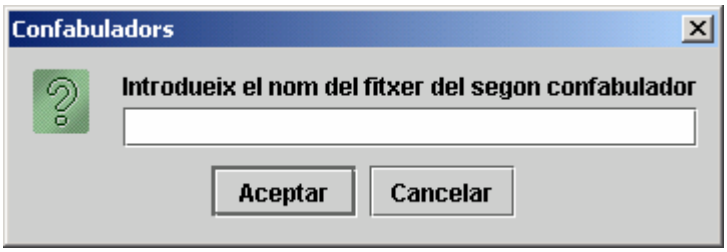
En el cas del que volem es crear una confabulació mitjançant dues imatges amb marca d'aigua diferent el que haurem de fer servir serà la instrucció *java Confabuladors*.



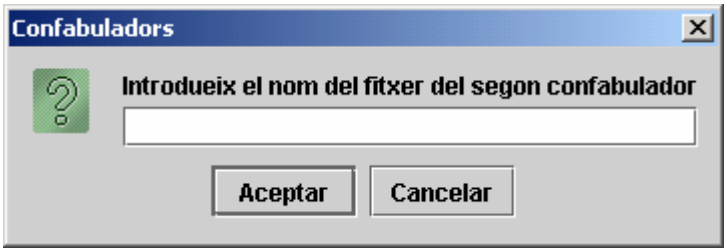
En la primera pantalla se'ns demana el nom del fitxer del primer confabulador.



I en la segona la del segon confabulador.



L'ultima pantalla ens demana el nom del fitxer on volem guardar la imatge resultant.



8. Bibliografia

Criptografia per als serveis telemàtics i el comerç electrònic. Josep Domingo Ferrer, Jordi Herrera Joancomartí. EDIUOC.

CODING AND INFORMATION THEORY. R .W.HAMMING. Prentice Hall, 1980.

http://www.iec.csic.es/criptonomicon/articulos/expertos64.html

http://www.iec.csic.es/criptonomicon/articulos/expertos34.html

http://www.watermarkingworld.org/

http://java.sun.com/j2se/1.4.2/docs/api/

http://www.encodedminds.com/archivos/estegano_en_bmp.txt

http://astronomy.swin.edu.au/~pbourke/dataformats/bmp/

http://www.javaworld.com/javaworld/javatips/jw-javatip43.html

http://java.sun.com/products/java-media/jai/

http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/index.html

http://java.sun.com/developer/onlineTraining/javaai/index.html

http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/

Graphics file formats : Reference and guide / C. Wayne Brown, Barry J. Shepherd. - Greenwich : Manning, cop. 1995. - XII, 472 p. ; 26 cm.

http://www.eccpage.com/

http://personales.mundivia.es/jtoledo/angel/error/error1.htm

http://www.cg.cs.tu-bs.de/v3d2/pubs/diwa-shg98.pdf

http://citeseer.ist.psu.edu/cache/papers/cs/14500/http:zSzzSzposeidon.csd.auth.grzSzpaperszSzPUBLISHEDzSzCONFERENCEzSzTsekeridou00azSzTsekeridou00a.pdf/wavelet-based-self-similar.pdf/

http://www.jjtc.com/Steganography/

http://www.jjtc.com/stegdoc/indexbk.html

Exploring Steganography:Seeing the Uncen. Neil F. Johnson, Sushil Jajodia George Mason University

http://www.engelschall.com/%7Esb/hamming/

http://web.syr.edu/~rrosenqu/ecc/linear/linear2.htm

http://worldcatlibraries.org/wcpa/ow/8600d9b3a44e97f3a19afeb4da09e526.html

http://ponton.dcs.fi.uva.es/web/teoria_informacion/codigos_java/modelo.html

http://www.eie.polyu.edu.hk/~enmzwang/adc/l-notes/node3.html

Xarxes de computadors I RBA Febrero 1999. Fundació per a la Universitat Oberta de Catalunya. Realizaciones Editoriales, S.L

http://jungla.dit.upm.es/~trdt/

http://directory.google.com/Top/Science/Math/Applications/Communication_Theory/Coding_Theory/?tc=1

http://www.ii.uib.no/~georg/coding/hyperpdf/hgs03aaecc.pdf

http://www.it.isy.liu.se/publikationer/indexeng.html#doktor

http://www.it.isy.liu.se/publikationer/LIU-TEK-THESIS-706.pdf

http://kison.uoc.edu/membres/jordi/pdf/

http://www.mth.msu.edu/~jhall/classes/codenotes/Hamming.pdf

http://citeseer.ist.psu.edu/cis?q=watermark&cs=1

http://www.csee.usf.edu/~smohanty/research/Reports/WMSurvey1999Mohanty.pdf

http://www.cs.cuhk.hk/~king/PUB/hkicc991.pdf

Digital Watermarking : A Tutorial Review. Saraju P. Mohanty _ Dept of Comp Sc and Eng. Unversity of South Florida Tampa, FL 33620

Fisiología de la conducta. Neil R. Carlson. Editorial Ariel S.A. (1999) N. R. Wagner

http://www.iec.csic.es/criptonomicon/articulos/expertos64.html

http://www.iec.csic.es/criptonomicon/articulos/expertos34.html

http://www.watermarkingworld.org/

9. Annexes.

9.1. Classe imatgeCopyright.

```
import java.awt.image.RenderedImage;
import java.awt.image.WritableRaster;
import java.awt.image.BufferedImage;

public class imatgeCopyright {

 public static void main(String[] args) {
  char opcio='0';
  while (opcio!='3'){
   try{
    String key="0";
    GUI panelInOut=new GUI();

    while (opcio!='1'&&opcio!='2'&&opcio!='3'){
     key=panelInOut.preguntarInfo("imatgeCopyright","1. Introduir Marca
d'aigua"+
        '\n'+"2. Llegir Marca d'aigua" +
        '\n'+"3. Sortir");
     opcio=key.charAt(0);
    }
    if (opcio!='3'){
     String filenameIn="";
     while (filenameIn.length()<1){
      filenameIn=panelInOut.preguntarInfo("imatgeCopyright","Introdueix
el nom del fitxer");
     }

     //creem un objecte per poder modificar la imatge
     RenderedImage image=utilimatge.loadImage(filenameIn);

     //creem un WritableRaster que ens servirà per copia la imatge i
modificar-la
     //primer li apliquem les propietats de la imatge (sampleModel,
tamany del dataBuffer)
     WritableRaster imagenIn=
image.getData().createCompatibleWritableRaster();

     //Introduim la imatge
     imagenIn.setRect(image.getData());


     if (opcio=='1'){

      byte[] marca = new byte[16];


      String codiclient="";
      while (codiclient.length()==0){

 codiclient=panelInOut.preguntarInfo("imatgeCopyright","Introdueix el
codi del client o deixal buit per crear un client nou");
       if (codiclient.length()==0){
        //creem un client nou
        String
nomClient=panelInOut.preguntarInfo("imatgeCopyright","Introdueix el
nom del nou client");
```

```
        codiclient=String.valueOf(bdclients.crearNouClient(nomClient));
       }

       //recuperem la marca de la base de dades que identifica al
client
       String
client=bdclients.recuperarMarca(Integer.parseInt(codiclient),marca);
       if (client=="") codiclient="";
      }


      //grabem la marca d'aigua en la imatge
      grabarmarca.marcar(marca, imagenIn);

      //la transformem en un objecte BufferedImage per poder-la grabar
      BufferedImage image2=new BufferedImage(image.getColorModel(),
imagenIn,false,null);

      //guardem la imatge modificada
      String filenameOut="";
      while (filenameOut.length()<1){

 filenameOut=panelInOut.preguntarInfo("imatgeCopyright","Introdueix el
nom de fitxer de destí");
      }

      //format en el que es guardarà la imatge
      String format = "bmp";

      //guardem la imatge
      RenderedImage imatgeOut=utilimatge.saveImage(filenameOut, format,
image2);
      opcio='0';

     }

     if (opcio=='2'){
      //llegim la marca
      byte[] marca2=llegirmarca.llegir(imagenIn);

      String marcaString=new String();
      for (int n=0;n<16;n++){
       marcaString=marcaString+(char)marca2[n];
      }
      String client=bdclients.recuperarClient(marca2);
      panelInOut.mostrarMissatge("marca d'aigua","La marca pertany a
'"+client+"'");
      opcio='0';
     }
    }
   }
   catch(Exception e){
    errors.error(e);
    opcio='0';
   }
  }
  System.exit(0);
 }
}
```

9.2. Classe grabarmarca.

```
import java.awt.image.WritableRaster;


public class grabarmarca {

 //numero de bits que podem enmaguetzemar a la imatge
 private static int NUMBITS=128;
 private static int colorsBanda=256;

 public static boolean[] extreureBits(byte[] codi){
  int n, n2;
  byte auxbyte;
  boolean bit;
  int longitud=(int)codi.length;
  boolean bits[]=new boolean[NUMBITS];
  for (n=0;n<longitud;n++){
   auxbyte=codi[n];
   for (n2=0;n2<8;n2++){
    if (((auxbyte>>(7-n2))&1)==1) bit=true;
    else bit=false;
    bits[(n*8)+n2]=bit;
   }
  }
  return bits;
 }

 public static void marcar(byte[] codi, WritableRaster imagenIn){
  boolean bits[];
  bits=extreureBits(codi);

  System.out.println("Introduïn la marca de agua...");
  //fem la modificació
  //primer obtenim el nombre de bandes
  int bandes=imagenIn.getNumBands();

  //i després les seves l'amplada i l'altura de la imatge
  int amplada=imagenIn.getWidth();
  int altura=imagenIn.getHeight();

  //modifiquem la imatge
  int color;
  int n,x,y;
  int bitnum;

  int offset=colorsBanda/bandes&510;//sera parell

  //modifiquem un a un tots els pixels de totes les bandes de la
imatge
  for (x=0;x<amplada;x++){
   for (y=0;y<altura;y++){
    for (n=0;n<bandes;n++){
     color=imagenIn.getSample(x,y,n);

     //cambiem el color del pixel si el color es parell
     if ((color&1)==0){
      // desplaçament segons la banda que sigui
      // aixo es necessari per evitar problemes
      // amb imatges molt oscures o molt clares.
```

```
      // Ens permet tenir copies de les dades
      // triplicades si tenim 3 bandes (24bits de colors)

      bitnum=color+(colorsBanda);
      bitnum=bitnum-offset*n;
      bitnum=(bitnum%(colorsBanda))/2;


      //comprobem si el el bit es un 0 o un 1, si es un 0 li canviem el
color
      if (!bits[bitnum]) color++;
     }
     imagenIn.setSample(x,y,n,color);
    }
   }
  }
  System.out.println("Marca d'agua introduïda");
 }

}
```

9.3. Classe llegirmarca.

```
import java.awt.image.WritableRaster;

public class llegirmarca {

 private static double coeficient=0.6f;
 private static int colorsBanda=256;
 private static int NUMBITS=128;
 public static byte[] llegir(WritableRaster imatgeIn){
  byte[] marca=new byte[(int)(NUMBITS/8)];

  int h=imatgeIn.getHeight();
  int w=imatgeIn.getWidth();
  int bandes=imatgeIn.getNumBands();
  int bit=0;
  int bytenum;
  byte nuevobyte;
  int offset=colorsBanda/bandes&510;
  int n2;
  //llegim tots els pixel per saber quina quantitat tenim de cada
color per banda
  int[][] histograma=new int[colorsBanda][bandes];
  for (int n=0; n<h; n++){
   for (n2=0;n2<w; n2++){
    for (int n3=0;n3<bandes; n3++){
     histograma[imatgeIn.getSample(n2,n,n3)][n3]++;
    }
   }
  }
  float h1,h2;
  int max,idmax;

  //llegim els espais en el histograma
  for (int n=0; n<NUMBITS; n++){
   h1=0;
   h2=0;
   //comprobem si es un 0 o un 1 mirant totes les bandes

   for (n2=0; n2<bandes; n2++){
    //sumem les les quantitats de color del mateix bit d'informació
    //i el següent color, en les bandes que tingui la imatge
    h1+=histograma[(n*2+(n2*offset))%(colorsBanda)][n2];
    h2+=histograma[((n*2)+1+(n2*offset))%(colorsBanda)][n2];

   }
   //si la diferencia de quantitat de color supera el coeficient
   //es un 1 sino un 0
   if (h2==0) bit=0; //evitem la divisio per 0
   else if ((h1/h2)>coeficient) bit=1;
   else bit=0;
   bytenum=((int)n/8);
   marca[bytenum]=(byte)((marca[bytenum]<<1)+bit);
  }
  return marca;
 }
}
```

9.4. Classe codidual.

```
public class codidual {

 //nombre màxim de bytes per a la codificació del codi d'usuari
 public static int MAXBYTESIDENTIFICADOR=2;

 //nom del fitxer que conte la matriu generadora
 private static String matriuFitxer="matrgen.icx";

 public static byte[] codificar(int codi){
  int idFitxer;
  int x,y;
  byte caracter=0;
  String dimensioMatriuAux="";
  idFitxer=IOFitxer.obrirFitxerLectura(matriuFitxer);
  byte[] codificacio=new byte[MAXBYTESIDENTIFICADOR];
  if (idFitxer>0){

   //obtenim la dimensio horitzontal de la matriu
   while (caracter!=10){
    caracter=IOFitxer.llegirByte(idFitxer);
    dimensioMatriuAux+=(char)caracter;
   }


 x=Integer.parseInt(dimensioMatriuAux.substring(0,dimensioMatriuAux.le
ngth()-2));

   caracter=0;
   //obenim la dimensio vertical de la matriu
   dimensioMatriuAux="";
   while (caracter!=10){
    caracter=IOFitxer.llegirByte(idFitxer);
    dimensioMatriuAux+=(char)caracter;
   }

 y=Integer.parseInt(dimensioMatriuAux.substring(0,dimensioMatriuAux.le
ngth()-2));

   //llegim la matriu
   byte[][] matriuGeneradora=new byte[y][x];
   for (int n2=0;n2<y;n2++){
    for (int n=0;n<x;n++){
     caracter=IOFitxer.llegirByte(idFitxer);
     if ((char)caracter=='1') matriuGeneradora[n2][n]=1;
     else matriuGeneradora[n2][n]=0;
    }

    //llegim el caracter de retorn i linea següent
    IOFitxer.llegirByte(idFitxer);
    IOFitxer.llegirByte(idFitxer);

   }

   IOFitxer.tancarFitxerLectura(idFitxer);

   //multipliquem la matriu per el codi començant pel final de la
primera columna
   int bit, totalbit;
```

```
   for (int n=0;n<x;n++){
    totalbit=0;
    for (int n2=y-1;n2>=0;n2--){
     bit=(codi>>(y-n2-1))&1;
     bit*=matriuGeneradora[n2][n];

     totalbit=(totalbit+bit) % 2;
    }

    //movem tots els bytes un bit a l'esquerra
    mouBitsEsquerra(codificacio, 1, MAXBYTESIDENTIFICADOR);

    //sumem el bit al byte de més a la dreta
    codificacio[MAXBYTESIDENTIFICADOR-1]+=totalbit;
   }
  }
  return codificacio;
 }

 //mous x bits cap a l'esquerra en un número representat per bytes
 //el desplaçament ha de ser inferior a 8
 private static void mouBitsEsquerra(byte[] codificacioArg, int
desplacament, int numBytesArg){
  if (numBytesArg!=1){
   //desplaçem els bits de desplaçament a l'altre byte

   //primer fem el desplaçament dels bits del següent byte
   mouBitsEsquerra(codificacioArg,desplacament,numBytesArg-1);

   //i després, li sumem els bits desplaçats
   codificacioArg[numBytesArg-2]+=codificacioArg[numBytesArg-1]>>(8-
desplacament);
   codificacioArg[numBytesArg-1]<<=desplacament;
  }
  else codificacioArg[0]<<=desplacament;
 }
}
```

9.5. Classe IOFitxer.

```
import java.io.FileInputStream;
import java.io.DataInputStream;
import java.io.FileOutputStream;
import java.io.DataOutputStream;

public class IOFitxer {
 private static FileInputStream[] fitxerDadesLectura=new
FileInputStream[256];
 private static DataInputStream[] dadesStreamLectura=new
DataInputStream[256];
 private static FileOutputStream[] fitxerDadesEscritura=new
FileOutputStream[256];
 private static DataOutputStream[] dadesStreamEscritura=new
DataOutputStream[256];
 private static String[] nomFitxer=new String[256];
 private static int identificador=0;

 //obrim el fitxer amb el nom fitxerDadesStringArg per a lectura
 public static int obrirFitxerLectura(String fitxerDadesStringArg){
  identificador++;
  try{
   fitxerDadesLectura[identificador]=new
FileInputStream(fitxerDadesStringArg);
   dadesStreamLectura[identificador]=new
DataInputStream(fitxerDadesLectura[identificador]);
   return identificador;
  }
  catch(Exception e){
   errors.error(e);
   identificador--;
   return -1;
  }
 }

 //tanquem fitxer amb identificador identifiacadorArg per a lectura
 public static int tancarFitxerLectura(int identificadorArg){
  try{
   dadesStreamLectura[identificadorArg].close();
   fitxerDadesLectura[identificadorArg].close();
   //si em pogut tancar returnem 0
   return 0;
  }
  catch(Exception e){
   errors.error(e);
   //no em pogut tancar el fitxer, tornem -1
   return -1;
  }
 }

// obrim el fitxer amb el nom fitxerDadesStringArg per a escritura
 public static int obrirFitxerEscritura(String fitxerDadesStringArg){
  identificador++;
  int idTempEsc=identificador;
  try{

   nomFitxer[idTempEsc]=fitxerDadesStringArg;
   int idTempLect=obrirFitxerLectura(nomFitxer[idTempEsc]);
```

```
   fitxerDadesEscritura[idTempEsc]=new
FileOutputStream(nomFitxer[idTempEsc]+".tmp");
   dadesStreamEscritura[idTempEsc]=new
DataOutputStream(fitxerDadesEscritura[idTempEsc]);

   if (idTempLect!=-1){
    //copiem les dades per a treballar amb un fitxer temporal
    copiarDades(idTempLect, idTempEsc);

    tancarFitxerLectura(idTempLect);
   }
   return idTempEsc;
  }
  catch(Exception e){
   errors.error(e);
   identificador--;
   return -1;
  }
 }

 //copiar dades dels fitxers per a crear temporals
 public static void copiarDades(int idTempLect, int idTempEsc){
  byte byteIn=0;
  while (byteIn!=-1){
   byteIn=llegirByte(idTempLect);
   if (byteIn!=-1) escriureByte(idTempEsc,byteIn);
  }
 }

 //tanquem fitxer amb identificador identifiacadorArg per a lectura
 public static int tancarFitxerEscritura(int identificadorArg){
  try{
   dadesStreamEscritura[identificadorArg].close();
   fitxerDadesEscritura[identificadorArg].close();

   //obrim el fitxer manualment
   identificador++;
   int idEscr=identificador;
   fitxerDadesEscritura[idEscr]=new
FileOutputStream(nomFitxer[identificadorArg]);
   dadesStreamEscritura[idEscr]=new
DataOutputStream(fitxerDadesEscritura[idEscr]);


   int idLect=obrirFitxerLectura(nomFitxer[identificadorArg]+".tmp");

   copiarDades(idLect, idEscr);

   tancarFitxerLectura(idLect);

   dadesStreamEscritura[idEscr].close();
   fitxerDadesEscritura[idEscr].close();



   //si no hi ha problemes returnem 0
   return 0;
  }
  catch(Exception e){
   errors.error(e);
   //no em pogut tancar el fitxer, tornem -1
```

```
   return -1;
  }
 }

 public static byte llegirByte(int identificadorArg){
  try{
   return dadesStreamLectura[identificadorArg].readByte();
  }
  catch(Exception e){
   errors.error(e);
   return -1;
  }
 }

 public static int llegirEnter(int identificadorArg){
  try{
   return dadesStreamLectura[identificadorArg].readInt();
  }
  catch(Exception e){
   errors.error(e);
   return -1;
  }
 }

 public static char llegirCaracter(int identificadorArg){
  try{
   return dadesStreamLectura[identificadorArg].readChar();
  }
  catch(Exception e){
   errors.error(e);
   return '\0';
  }
 }

 public static int escriureEnter(int identificadorArg, int enterArg){
  try{
   dadesStreamEscritura[identificadorArg].writeInt(enterArg);
   return 0;
  }
  catch(Exception e){
   errors.error(e);
   return -1;
  }
 }

 public static int escriureCaracters(int identificadorArg, char
caracterArg){
  try{
   dadesStreamEscritura[identificadorArg].writeChar(caracterArg);
   return 0;
  }
  catch(Exception e){
   errors.error(e);
   return -1;
  }
 }

 public static int escriureString(int identificadorArg, String
caracterArg){
  try{
   dadesStreamEscritura[identificadorArg].writeChars(caracterArg);
```

```
   return 0;
  }
  catch(Exception e){
   errors.error(e);
   return -1;
  }
 }


 public static int escriureBytes(int identificadorArg, byte[]
byteArg){
  try{

 dadesStreamEscritura[identificadorArg].write(byteArg,0,byteArg.length
);
   return 0;
  }
  catch(Exception e){
   errors.error(e);
   return -1;
  }
 }

 public static int escriureByte(int identificadorArg, byte byteArg){
  try{
   dadesStreamEscritura[identificadorArg].write(byteArg);
   return 0;
  }
  catch(Exception e){
   errors.error(e);
   return -1;
  }
 }
}
```

9.6. Classe bdclients.

```
public class bdclients {

 //nombre màxim de clients, sempre inferior a 65536
 public static int MAXCLIENTS=8;

 //nombre màxim de bytes per a la codificació del client
 public static int
MAXBYTESIDENTIFICADOR=codidual.MAXBYTESIDENTIFICADOR;


 private static String fitxerDadesString="dadesClients.icd";
 private static String registre;
 private static int regCodiClient;
 private static byte[] regMarca=new byte[MAXBYTESIDENTIFICADOR];
 private static String regNomClient;

 public static int crearNouClient(String nomClientArg) {
  int codiClient=treuNouCodiClient();
  byte[] marca=codificar(codiClient);
  if (insertarNouClient(codiClient, marca, nomClientArg)) return
codiClient;
  else return -1;
 }

 //insertem un client nou
 private static boolean insertarNouClient(int codiClientArg, byte[]
marcaArg, String nomClientArg){
  int idFitxer;
  idFitxer=IOFitxer.obrirFitxerEscritura(fitxerDadesString);
  if (idFitxer>0){
   IOFitxer.escriureEnter(idFitxer, codiClientArg);
   IOFitxer.escriureBytes(idFitxer, marcaArg);
   IOFitxer.escriureString(idFitxer, nomClientArg);
   IOFitxer.escriureCaracters(idFitxer, '\n');
  }
  IOFitxer.tancarFitxerEscritura(idFitxer);
  return true;
 }

 //recuperem la marca asignada al client amb codi codiClientArg de la
base de dades
 public static String recuperarMarca(int codiClientArg, byte[]
marcaArg){
  int idFitxer;
  String client="";
  idFitxer=IOFitxer.obrirFitxerLectura(fitxerDadesString);
  if (idFitxer>0){
   while (seguentRegistre(idFitxer)&&(client.length()==0)){
    if (regCodiClient==codiClientArg){
     client=regNomClient.toString();
     copiarMarca(regMarca, marcaArg);
    }
   }
  }
  IOFitxer.tancarFitxerLectura(idFitxer);
  return client;
 }
```

```
 //recuperem el client asignat a la marca de la base de dades
 public static String recuperarClient(byte[] marcaArg){
  int idFitxer;
  int resultat;
  int total;
  int bitactual;
  int maxim=0;
  String client="";
  idFitxer=IOFitxer.obrirFitxerLectura(fitxerDadesString);
  if (idFitxer>0){
   while (seguentRegistre(idFitxer)){
    total=0;
    for (int n=0; n<regMarca.length;n++){
     //fem XOR bit a bit, si es 0 no hi ha diferencia, si es 1 aumentem
distancia hamming

     resultat=regMarca[regMarca.length-n-1]^marcaArg[marcaArg.length-n-
1];

     //contem els 0 que indiquen que son iguals les dues marques
     for (int bits=0;bits<8;bits++){
      bitactual=(resultat>>bits)&1;
      if (bitactual==0) total++;
     }
    }
    if (total>maxim){
     client=regNomClient.toString();
     maxim=total;
    }
   }
  }
  IOFitxer.tancarFitxerLectura(idFitxer);
  return client;
 }

 //copiem la els bytes que representan la marca en altre grup de bytes
 private static void copiarMarca(byte[] marcaOrg, byte[] marcaDest){
  for (int n=0;n<marcaDest.length;n++){
   marcaDest[n]=0;
  }
  for (int n=0;n<marcaOrg.length;n++){

   marcaDest[marcaDest.length-n-1]=marcaOrg[marcaOrg.length-n-1];
  }
 }

 //obtenim un codi de client nou
 private static int treuNouCodiClient() {
  int codiClient;
  int idFitxer;
  regCodiClient=0;
  idFitxer=IOFitxer.obrirFitxerLectura(fitxerDadesString);
  if (idFitxer==-1){
   //creem el fitxer si no existeix
   idFitxer=IOFitxer.obrirFitxerEscritura(fitxerDadesString);
   IOFitxer.tancarFitxerEscritura(idFitxer);

   idFitxer=IOFitxer.obrirFitxerLectura(fitxerDadesString);
  }
  if (idFitxer>0){
   codiClient=0;
```

```
   do{
    codiClient=regCodiClient;
   } while (seguentRegistre(idFitxer));

   IOFitxer.tancarFitxerLectura(idFitxer);
   codiClient++;
   return codiClient;
  }
  else return -1;
 }

 //codifiquem el codi client
 private static byte[] codificar(int CodiClientArg){
  return codidual.codificar(CodiClientArg);
 }

 private static boolean seguentRegistre(int idFitxerArg){
  //intentem llegir les dades del fitxer de clients
  try{
   regCodiClient=IOFitxer.llegirEnter(idFitxerArg);
   //si el fitxer esta buit surtim
   if (regCodiClient==-1) return false;
   for (int n=0;n<MAXBYTESIDENTIFICADOR;n++){
    regMarca[n]=IOFitxer.llegirByte(idFitxerArg);
   }
   regNomClient="";
   char caracter='\r';
   while ((caracter!='\n')&&(caracter!='\0')){
    caracter=IOFitxer.llegirCaracter(idFitxerArg);
    regNomClient+=caracter;
   }
  }
  catch(Exception e){
   errors.error(e);

   //es el final del fitxer
   return false;
  }
  return true;
 }

}
```

9.7. Classe Confabuladors.

```
import java.awt.image.RenderedImage;
import java.awt.image.WritableRaster;
import java.awt.image.BufferedImage;
import java.lang.Math;

public class Confabuladors {

 public static void main(String[] args) {
  GUI panelInOut=new GUI();
  String conf1=panelInOut.preguntarInfo("Confabuladors","Introdueix el
nom del fitxer del primer confabulador");
  String conf2=panelInOut.preguntarInfo("Confabuladors","Introdueix el
nom del fitxer del segon confabulador");
  String desti=panelInOut.preguntarInfo("Confabuladors","Introdueix el
nom del fitxer desti");

  //creem un objecte per poder treballar amb les imatges
  RenderedImage imageConf1=utilimatge.loadImage(conf1);
  RenderedImage imageConf2=utilimatge.loadImage(conf2);

  //creem un WritableRaster que ens servirà per copia la imatge i
modificar-la
  //primer li apliquem les propietats de la imatge (sampleModel,
tamany del dataBuffer)
  WritableRaster imageOrg=
imageConf1.getData().createCompatibleWritableRaster();
  WritableRaster imageDes=
imageConf2.getData().createCompatibleWritableRaster();

  //Introduim la imatge
  imageOrg.setRect(imageConf1.getData());
  imageDes.setRect(imageConf2.getData());


  //obtenim les dades de la imatge
  int h=imageOrg.getHeight();
  int w=imageOrg.getWidth();
  int bandes=imageOrg.getNumBands();

  int color1;
  int color2;


  //comprovem pixel a pixel si les imatges son diferents i posem un
del
  //dos bits si aleatoriament
  for (int x=0;x<w;x++){
   for (int y=0;y<h;y++){
    for (int banda=0;banda<bandes;banda++){
     color1=imageOrg.getSample(x,y,banda);
     color2=imageDes.getSample(x,y,banda);
     if (color1!=color2){
      if(Math.random()>0.5){
       imageDes.setSample(x,y,banda,color1);
      }
      else imageDes.setSample(x,y,banda,color2);
     }
    }
```

```
   }
  }


  //la transformem en un objecte BufferedImage per poder-la grabar
  BufferedImage image=new BufferedImage(imageConf2.getColorModel(),
imageDes,false,null);

  //la grabem
  String format = "bmp";
  RenderedImage imatgeOut=utilimatge.saveImage(desti, format, image);

  System.exit(0);
 }
}
```

9.8. Classe utilimatge.

```
import javax.media.jai.JAI;
import java.awt.image.RenderedImage;

public class utilimatge {

 public static RenderedImage loadImage(String filenameIn){
 //carreguem el fitexer
 RenderedImage image=null;
 try{
  System.out.println("Carregan fitxer "+filenameIn);
  image = JAI.create("fileload", filenameIn);
 }
 catch(Exception e){
  errors.error(e);
 }
 finally{
  return image;
 }
}

public static RenderedImage saveImage(String filenameOut, String
format, RenderedImage imageIn){
 //guardem la imatge en el format especificat
 RenderedImage image=null;
 try{
  System.out.println("Guardan fitxer "+filenameOut);
  image = JAI.create("filestore", imageIn, filenameOut, format);
  System.out.println("Fitxer guardat correctament");
 }
 catch(Exception e){
  errors.error(e);
 }
 finally{
  return image;
 }
}
}
```

9.9. Classe GUI.

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import java.awt.Window;

public class GUI{
 JFrame frame;

 public String preguntarInfo(String titol, String msg){
  return JOptionPane.showInputDialog(frame, msg, titol,
JOptionPane.QUESTION_MESSAGE);
   }

   public void mostrarMissatge(String titol, String msg){
  JOptionPane.showMessageDialog(frame, msg, titol,
JOptionPane.INFORMATION_MESSAGE);
   }

 public GUI() {
  frame = new JFrame();
  Window w = new Window(frame);
  w.setLocation(10, 10);
 }

}
```

9.10. Classe errors.

```
public class errors {

 public static void error(Exception e){
  System.out.println(e.toString());
 }
}
```