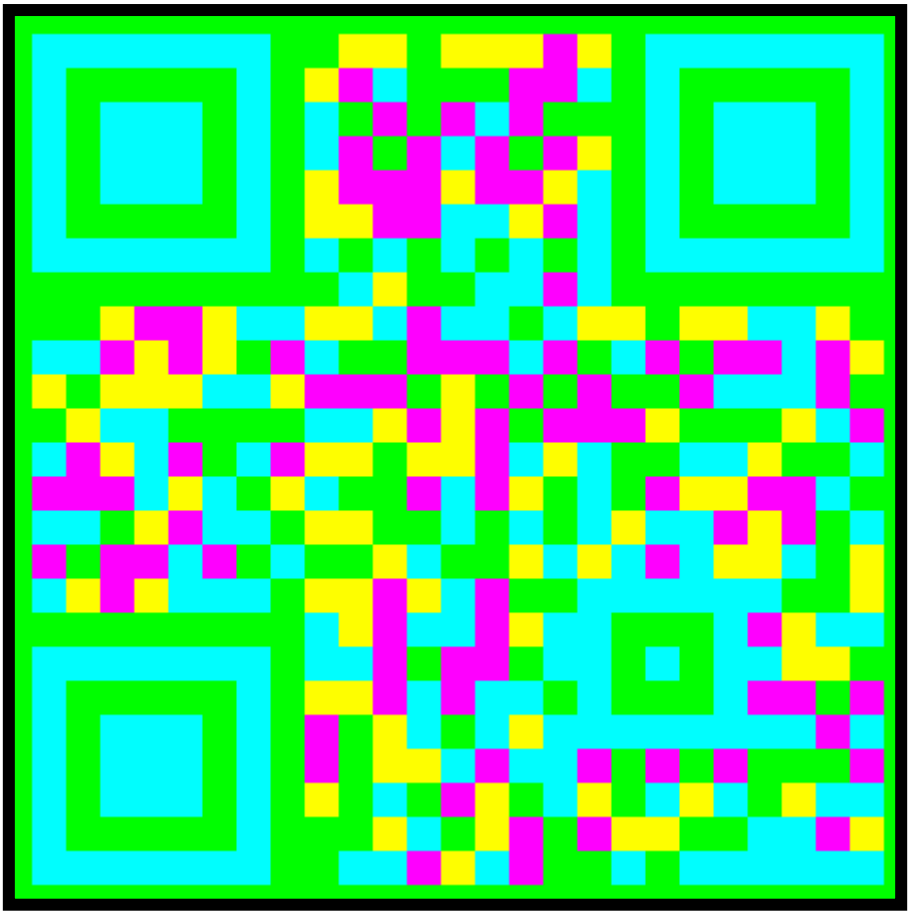
JAVA I CODIS QR



ALUMNE: ARSLAAN-HAMZA MOHAMMAD JABEEN

DIRIGIT PER: JAVIER CIVIT NOGUERA

2n BATXILLERAT

IES VERDAGUER

BARCELONA GENER 20011-2012

NOTA D'AGRAÏMENTS

Abans de començar, m'agradaria agrair a un seguit de persones que m'han ajudat a realitzar aquest treball:

- Al meu tutor de recerca per haver tingut paciència amb mi i haver-me ajudat en els moments en que estava perdut.
- A la meva germana Laareb, sense la qual el disseny d'aquest treball restaria ensopit.
- A tota la meva família, amics i tots aquells que han fet possible la realització d'aquest treball

ÍNDEX DE TREBALL

PÀGINA

1. INTRODUCCIÓ..... 5

2. UNA MICA D'HISTÒRIA 7

3. CODIS QR 8

3.1 DEFINICIONS..... 8

3.2 EL CODI QR..... 9

3.3 CARACTERÍSTIQUES 10

3.3.1 MODE 10

3.3.2 CODEWORDS 11

3.3.3 PATRONS 11

3.3.3.1 ARITMÈTICA MODULAR 12

4. JAVA..... 14

4.1 COMENÇANT A PROGRAMAR EN JAVA 14

4.1.1 EL MÈTODE MAIN..... 14

4.1.2 VARIABLES..... 16

4.1.3 ESTRUCTURES DE REPETICIÓ I OPERADORS 17

4.1.4 PROGRAMACIÓ ORIENTADA A OBJECTES 19

4.1.5 MÈTODES 21

4.1.6 LLIBRERIES 21

4.1.7 LA CLASSE JFRAME 22

4.1.8 LA CLASSE JPANEL..... 24

4.1.9 TRACTAMENT D'IMATGES..... 25

5. COM ES PROGRAMA UN PROGRAMA CODIFICADOR DE CODIS QR 27

5.1 CODIFICACIÓ D'UN CODI QR AMB LA LLIBRERIA ZXING..... 27

5.2 DIAGRAMA DE CODIFICACIÓ..... 32

5.3 ESQUEMA VISUAL..... 33

5.4 PROGRAMA CODIFICADOR DE CODIS QR 33

6. COM ES PROGRAMA UN PROGRAMA DECODIFICADOR DE CODIS QR... 46

6.1 DIAGRAMA DE DESCODIFICACIÓ 46

6.2 ESQUEMA VISUAL 47

6.3 PROGRAMA DESCODIFICADOR DE CODIS QR 48

7. HIPÒTESI PER AUGMENTAR LA MEMÒRIA D'UN CODI QR 54

7.1 DISCUTIM LES HIPOTESIS 54

8. PROGRAMA CREADOR DE CODIS QR AMB CAPACITAT D'EMMAGATZEMATGE AUGMENTADA 55

9. PROGRAMA DESCODIFICADOR DE CODIS QR MODIFICATS 57

10. CONCLUSIONS..... 58

11. ANNEX 59

12. BIBLIOGRAFIA..... 59

1. INTRODUCCIÓ

Aquesta recerca no hauria començat si la meva professora de matemàtiques de 1er de Batxillerat, Marisol Gimeno, no m'hagués proposat fer el meu Treball de Recerca sobre l'aritmètica modular aplicada a codis de barres. Però encara dubtava, volia fer alguna cosa innovadora, que pogués ser utilitzada per tothom i que a més representés un avenç per minúscul que fos. La idea de que el meu Treball de Recerca restés com un com un conjunt d'informació extreta d'Internet no m'agradava. Volia un treball que m'obligués a aprendre coses que no sabia, que em motivés a fer recerca.

Quan buscava informació relacionada amb l'aritmètica modular en el fons no parava de pensar en com seria programar programes i va ser quan vaig aprofundir en la recerca d'informació sobre codis de barres quan vaig estar segur que volia fer un programa relacionat amb els codis QR. Em va fascinar la idea de transformar un text escrit a un codi que podria tornar a ser llegible mitjançant un programa de descodificació.

A començaments de les vacances d'estiu tenia clar que per dur a terme la meva recerca hauria d'aprendre un llenguatge de programació i vaig recordar que Marisol em va recomanar JAVA. Vaig llogar llibres de diverses biblioteques de Barcelona i vaig començar a llegir-los. Al començament em costava molt entendre'ls i se'm va ocorre buscar tutorials en Internet dirigits per a persones que s'iniciaven a la programació. Em van agradar molt un seguit de capítols audiovisuals de programació per JAVA (veure bibliografia per més informació), que conjuntament amb els llibres de la biblioteca em van facilitar molt l'aprenentatge d'aquest llenguatge.

Quan em va semblar que tenia el material suficient per a realitzar els programes, vaig aprofundir sobre el codis QR i em vaig topar amb 2 treballs de fi de carrera dedicats expressament per ells. Una vegada après el que són i com s'estructuren

vaig voler donar un pas més. No estaria del tot satisfet personalment si només hagués realitzat un treball sobre codis QR, per aquesta raó vaig voler desenvolupar les meves pròpies hipòtesis, volia augmentar la capacitat d'emmagatzematge dels codis QR i és al final d'aquest treball en el qual intentaré exposar una nova visió pels codis de barres.

2.UNA MICA D'HISTÒRIA

És imprescindible fer una ullada als codis de barres anteriors al codi QR per entendre millor aquest concepte. L'aplicació d'aquests ha anat variant al llarg de la història però no el seu objectiu, que va ser i és codificar informació en un codi que permeti la seva lectura i descodificació.

La idea d'un codi de barres va sorgir l'any 1932 en el qual Wallace Flint va proposar la idea de gestió de l'inventari d'un supermercat utilitzant targetes perforades per identificar el que s'ha adquirit. El primer codi de barres va aprèixer l'any 1949 creat per Norman J.Woodland, Bernard Silver i Jordin Johanson a partir de cercles concèntrics que era llegit per un fotodetector. Van presentar la patent al mateix any però no va ser fins l'any 1952 que se'ls va concedir. Posteriorment van treballar en la tecnologia que s'encarregaria de interactuar amb aquest codi però no van poder crear-ne un lector suficientment fiable perquè en aquell moment no contaven amb les eines per fer-ho. L'any 1962 l'empresa Philco (pionera en fabricar radios, televisions i bateries) va comprar la seva patent.

El 1961 l'emprese Silvana General Telephone va instal·lar el primer escàner fixe de codis de barres de tinta de color vermell, blau i blanc. El projecte va fracassar a causa de la manca de manteniment de les etiquetes del codi.

L'any 1969 es troba la primera aplicació industrial del làser(es fa servir per soldadures en automòvils) i també una eina que acabaria sent fonamental per la descodificació dels codis de barres.Un any després l'UGPIC (Universal Grocery Products Identification Code) crearen un comitè per seleccionar un codi de barres estàndard. Tres anys després es selecciona el codi UPC per la venta de productes, codi que va néixer per una proposta d'IBM. El 1979 s'aprova el codi EAN que és la versió Europea de l'UPC.

A l'any 1980 apareix PostNet, una codificació utilitzada pels correus d'Estats Units, i un any després el codi 128, el primer codi alfanumèric.

3. EL CODI QR

3.1 DEFINICIONS

En aquest apartat apareixen termes que són de difícil comprensió si prèviament no es coneix la seva definició, per aquesta raó s'exposen les definicions abans de començar la explicació dels codis QR.

Bits de farcit: Conjunt de bits que s'utilitzen per omplir l'últim codeword després d'una cadena de codewords inacabada.

Bits restants: Bits utilitzats per omplir la regió de codificació on l'espai no es divideix exactament en 8 bits.

Codeword: Conjunt de 8 bits que conté la informació del codi QR. Aquest conjunt pot tenir diferents formes al llarg del codi.

Comptador de caràcters: Conjunt de bits que identifiquen el nombre de caràcters que conté una cadena de dades en un mode determinat.

Emmascarar dades: Operació que es realitza als bits de dades per augmentar la diferenciació entre mòduls blancs i negres amb la finalitat d'una millor descodificació.

Identificador de mode: Regió del codi QR que conté la informació del mode en que està codificat el codi QR.

Informació de format: Regió del codi QR que conté la informació relacionada amb el patró de mascara aplicat al codi i el nivell de correcció d'errors d'aquest.

Informació de la versió: Regió de codi exclusiva per als codis de versió 7 o superior que indica la versió del símbol.

Mode: Forma en que es representa un tipus de caràcter (kanji, numèric, alfanumèric) en una cadena de bits

Mòdul: Llindar blanc o negre d'un codi QR.

Patró d'alineament: Patró que s'utilitza en la descodificació del codi per reposicionar la matriu per la seva correcta descodificació.

Patró localitzador: Patró situat en la cantonada superior dreta, esquerra i inferior esquerra que permet la identificació del codi.

Regió de descodificació: Regió ocupada per codewords de dades, de correció d'errors, d'informació i de format.

Separador: Seqüència de bits 0 d'amplada 1 que separa els patrons localitzadors de la resta de codi.

Terminador: Seqüència de bits 0 que s'introdueixen al final de la cadena de bits per identificar el final del codi

Versió: Indica la quantitat de informació que un codi pot emmagatzemar. La versió varia de la 1 que n' és la que menys informació pot emmagatzemar i la 40 la que més.

Zona de silenci (Quiet Zone): Conjunt d'espai en blanc que rodeja el codi que té mínim 4 bits d'amplitud.

3.2 EL CODI QR

El codi QR va ser creat l'any 1994 per la companyia japonesa Denso-Wave que posteriorment va decidir distribuir-lo lliurement sense aplicar els drets de la seva patent. L'any 2000 la ISO (International Organization for Standardization) conjuntamen amb la IEC(International Electrotechincal Comission) crearen un document oficial en anglés en el qual s'especificaven totes les característiques d'aquest codi. Aquest document va ser revisat anys després i la seva última revisió n'és la versió ISO/IEC 18004/2006 amb el títol: "Information technology-Automatic technology and data capture techniques". Desgraciadament el document compta amb copyright i es ven només des d' aquestes dues organitzacions, però l'any 1999 la JIS(Japanase Industrial Standards) va crear un document de finalitat semblant i lliure de copyright. La seva última versió és la de 2004.

3.3 CARACTERÍSTIQUES

El codi QR és un conjunt de mòduls distribuïts en l'espai que representen un 1 o un 0 .Consta de 40 versions diferents que depenen directament de la mida d'aquest, la versió 1 és de 21 x 21 mòduls i la versió 40 n'és de 177x177 mòduls.

3.3.1 MODE

Els diversos modes de codificació permeten aprofitar millor l'espai. Aquests es representen en la regió exclusiva per la informació de format del codi QR.

- El mode numèric(0-9) permet codificar com a màxim 7089 números en un mateix codi , generalment cada 3 números a codificar es generen 10 bits.

- El mode alfanumèric (a-z,0-9) permet codificar com a màxim 4296 en el mateix codi. Per a fer-ho primer a cada caràcter se li assigna un valor entre 0 i 44, i és aquest valor que es converteix a bits per ser representats.

Char.	Valor	Char.	Valor	Char.	Valor	Char.	Valor	Char.	Valor	Char.	Valor	Char.	Valor	Char.	Valor
0	0	6	6	C	12	I	18	O	24	U	30	SP	36	.	42
1	1	7	7	D	13	J	19	P	25	V	31	$	37	/	43
2	2	8	8	E	14	K	20	Q	26	W	32	%	38	:	44
3	3	9	9	F	15	L	21	R	27	X	33	*	39		
4	4	A	10	G	16	M	22	S	28	Y	34	+	40		
5	5	B	11	H	17	N	23	T	29	Z	35	-	41		

- El mode Byte permet codificar 2953 Bytes. Aquest és el mode més fàcil de codificació perquè no necessita ser convertit per a ser representat, és representa cada byte en un codeword de 8 bits.

- El mode Kanji(japonès) permet codificar com a màxim 1817 caràcters.

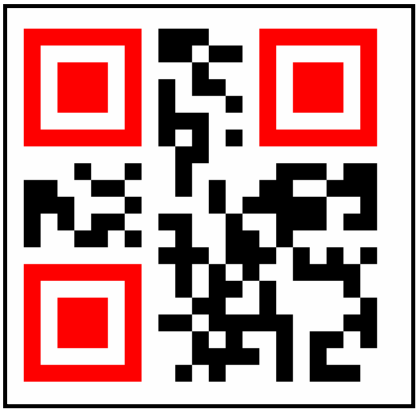
3.3.2 CODEWORDS

La informació de dintre del codi està dividida en codewords. Els codewords són conjunts de 8 bits que varien la seva forma al llarg del codi. Aquests codewords poden ser de 2 tipus: d'informació i de correcció d'errors. Els codewords d'informació, com el seu nom indica, contenen estrictament la informació transformada en bits. A si mateix, els codewords de correcció d'errors contenen la informació necessària per la reconstrucció d'un codi deteriorat per agents externs. Existeixen 2 tipus d'agressions externes que deterioren el codi: les que causen un error (un mòdul blanc que es converteix en un de negre) i les que no permeten llegir el mòdul (per exemple que la imatge no estigués enfocada correctament). La restauració del codi depèn de l'agressió externa, per es podran corregir més o menys errors segons si és un error o un borró (veure annex per més informació). Depenent de l'usuari, a l'hora de codificar el codi es pot elegir entre 4 nivells de correcció d'errors

Capacitat correctora d'errors del QR-Code		
Nivell L	(baix)	7 %
Nivell M	(mitjà)	15 %
Nivell Q	(mitjà-alt)	25 %
Nivell H	(alt)	30 %

3.3.3 PATRONS

Els patrons localitzador el fan únic i permeten la seva descodificació des de qualsevol angle. És a dir, si la imatge és captada des d'un angle qualsevol aquesta es pot modificar de tal manera que el codi pugui posar-se en la posició correcte sense que la reconstrucció afecti a la informació que conté.



Hi ha tres per totes les versions del símbol, un situat en la cantonada superior esquerre, un altre en la cantonada superior dreta i un altra en la

cantonada inferior esquerre. Cadascun està format per un quadrat negre format per 3 x 3 bits rodejat per 5 x 5 mòduls blancs i aquest rodejats a la seva vegada per 7 x 7 mòduls negres.

Els patrons temporitzadors s'encarreguen d'ajudar a determinar les coordenades dels mòduls i del símbol. Aquests estan formats per una línea vertical i una d'horitzontal de mòduls blancs i negres intercalats que comencen i acaben en un mòdul negre. El patró temporitzador vertical, esta situat en la columna 9 i comprèn des del patró localitzador superior esquerre fins al patró localitzador inferior esquerre. El patró temporitzador horitzontal està situat en la fila 9 i comprèn des del patró localitzador superior esquerre fins al patró localitzador superior dret.

Els últims 3 bits de la informació del format es dediquen per determinar el patró de màscara que s'aplicarà a l'hora de codificar. Aquest pas consisteix en aplicar una condició a l'hora d'introduir els bits en el codi i els bits que compleixin la condició es representaran. És útil per garantir una millor descodificació ja que els bits blancs i negres s'intercalen i es redueix la possibilitat de trobar-nos amb un error. La condició introduïda està basada en aritmètica modular. En el següent apartat farem un incís per explicar el que és .

3.3.3.1 ARITMETICA MODULAR

Per a definir l'aritmètica modular es important tenir clar el concepte de congruències. Les congruències són relacions que es donen entre 2 nombres enters (a , b) que comparteixen un mateix residu al ser dividits entre un nombre enter (c). Per tant podríem expressar aquesta relació de la següent forma:

$$a \equiv b \ (mod\ c\ )$$

L'aritmètica modular, doncs, és l'estudi d'aquestes relacions i les operacions elementals fetes amb aquestes.

SUMA

Al sumar un nombre en aquest tipus d'aritmètica, primer es fa la suma normal i posteriorment es calcula el residu de la suma.

Exemple:

$$9 + 4 \equiv 3 \ (mod\ 5)$$

PRODUCTE

Pel càlcul del producte el mètode és el mateix que l'anterior, primer es calcula el producte i després el residu d'aquest.

Exemple:

$$10 \cdot 2 \equiv 1 \ (mod\ 19)$$

Es pot observar que la suma i el producte en Z/m (aritmètica mòdul m) presenten unes propietats:

$$Sigui\ m, a, b, c, d \ \ nombres\ enters\ tals\ que\ a \equiv b \ (mod\ m)\ i\ c \equiv d \ (mod\ m)$$

$$a + c \equiv b + d \ (mod\ m)$$

$$a \cdot c \equiv b \cdot d \ (\ mod\ m)$$

Aquest concepte és la base de les demostracions de tots els criteris de divisibilitat de nombres enters.

4. JAVA

Java és un llenguatge de programació creat per l'empresa Sun Microsystems i se sembla superficialment al llenguatge C i C++. James Gosling, un dels pares de Java, a principis dels anys 80 va crear un sistema de finestres pel control d'interfícies gràfiques en un escriptori denominat NeWS. Aquest sistema va perdurar durant un breu temps competint amb X Window però no va poder fer front a aquest gegant i va caure en l'oblit. Posteriorment Gosling va treballar en un projecte que portaria a la creació de la filial de Sun, First Person. Aquesta filial va començar dissenyant programes per a aparells com telèfons mòbils i ordinadors de butxaca pels que tocava fer programes petits, segurs i senzills que consumissin poca memòria i es van adonar que el llenguatge C++ ja no era l'adequat per a ells i per a aquesta raó Gosling es va a posar a treballar en un llenguatge que denominaria "C++ minus minus". Per culpa del poc interès que mostrava la PDA (personal digital assistant) Apple Newton envers al públic, First Person va centrar la seva atenció en crear un llenguatge de programació per a televisions interactives que denominaria Oak. Oak no va trobar moltes sortides pel poc interès que mostrava la gent envers aquest tipus de televisions i aviat es va deixar de treballar en aquest projecte. Va ser en 1993, aprofitant el boom d'Internet, quan Gosling i Bill Joy, director d'investigació de Sun Microsystems, van crear un llenguatge de programació especialment per la xarxa que aprofitaria els coneixements de NeWS, c++minus minus i Oak batejat com JAVA.

4.1 COMENÇANT A PROGRAMAR EN JAVA

En aquest apartat intentaré exposar les parts més importants per l'aprenentatge d'aquest llenguatge.

4.1.1 EL MÈTODE MAIN

El mètode main és el primer mètode que busca java per a executar un programa i, per tant sempre està present en algun lloc del codi del programa. Aquest mètode pot establir-se dintre d'una classe o en una classe especialment per a ell, es

recomana per temes d'organització crear una classe especialment per a ell. Primer de tot, com en tota classe, s'ha de donar nom a la classe. Aquí es pot posar qualsevol nom (excepte main en minúscules) però ha d' estar seguit d'una clau. Aquesta clau serveix per identificar l'inici i el final de cada classe, per tant al final de la classe ha d'haver-hi un altra clau que la tanqui. Després s'ha d'escriure un seguit de codi que més endavant s'explicarà què significa. Finalment el codi resultant ha de quedar així:

```
public class Main {

    public static void main(String[] args) {

    }

}
```

Per introduir les comandes i fer que s'executin només necessitarem escriure-les entre el tros que abarca el mètode main;

```
public class Main {

    public static void main(String[] args) {
        System.out.print("Hola món");
    }

}
```

I la resposta de l'ordinador serà:

Amb la comanda " System.out.print()" fem que l'ordinador mostri el text o nombres que introduïm entre els parèntesi. Com podem observar a l'hora d'introduir el text entre el parèntesi de la comanda s'ha fet entre cometes. Aquestes cometes s'utilitzen per a què l'ordinador pugui identificar que la informació que s'està introduint són lletres o Strings, més endavant s'explicarà què són els Strings. A l'acabar d'escriure cada comanda es posa un punt i coma per a què la màquina pugui identificar el final de la comanda.

4.1.2 VARIABLES

En java la utilització de variables és un procés molt comú i senzill, però per fer-ho primer s'ha d'especificar quin tipus de variable es crea. Per tant primer de tot s'especifica que la variable és un número enter (int), unes lletres(String), bytes (byte), un nombre amb comes (double) un número molt llarg(long) o un objecte(object). Aquestes distincions s'han de fer obligatòriament per a què l'ordinador pugui distingir quin tipus d'informació està manegant. Després s'especifica el nom de la variable, i finalment quin valor té la variable. A l'especificar el valor no fa falta escriure res més que el valor en si però quan és un String, obligatòriament s'ha de fer-ho entre cometes. Aquesta particularitat present en més d'un llenguatge de programació es deu a la necessitat de l'ordinador de no confondre els números que s'expressen com lletres a nombres del tipus int o double.

```
public class Main {

    public static void main(String[] args) {

        int a=786;
        double b=7.86;
        String c="Cotxe";

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);


    }

}
```

El que mostra la consola:

```
<terminated> Main (15) [Java Application] E:\Archivos de programa\Java\jre6\bin\javaw.exe (19/09/2011 00:09:49)
786
7.86
Cotxe
```

*Aquesta vegada s'ha fet ús de la comanda "System.out.println()" enlloc de System.out . print() perquè entre comanda i comanda automàticament s'estableix un salt de línea.

Amb aquestes variables es poden fer sumes, restes, multiplicacions,divisions tenint en compte que els membres de cada operació han de ser del mateix tipus. A més

s'ha de destacar que la divisió de 2 membres del tipus int dóna com a quocient un nombre enter, és a dir, l'ordinador aproxima al nombre enter més petit i a la vegada més proper al nombre real.

```
1
2  public class Main {
3
4      public static void main(String[] args) {
5
6          int a=18;     // amb la doble barra es poden introduir comentaris
7          a=a+1;     //Aqui indiquem que la variable a es el valor de a + 1 ,
8          System.out.println(a);    // Comprovem el resultat mostrant-lo per pantalla
9          a=a/4;     // Dividim entre 4
10         System.out.println(a);    //Tornem a comprovar
11
12
13     }
14
15 }
```

Resultat:

```
<terminated> Main (15) [Java Application] E:\Archivos de programa\Java\jre6\bin\javaw.exe (19/09/2011 03:31:25)
19
4
```

4.1.3 ESTRUCTURES DE REPETICIÓ I OPERADORS LÒGICS

Una estructura de repetició, com el nom diu, és un segment de codi que es repeteix durant un cert nombre de vegades. Hi ha diferents tipus d'estructures: while, if,else, for, i switch.

WHILE:

En català "mentre", és una comanda que repeteix un tros de codi fins que la condició que li posem sigui certa. Per a indicar quantes vegades s'ha de repetir el codi s'utilitzen operadors lògics com el **<** (més petit que) ,**<=**(més petit o igual que),**>**(més gran que),**>=**(més gran o igual que), **!=** (diferent que), **==**(igual que). Per exemple :

```
public class Main {

    public static void main(String[] args) {

        int a=18;
        while(a<23){
            a=a+1;
            System.out.println(a);
        }

    }

}
```

Resultat:

```
<terminated> Main (15) [Java Application] E:\Archivos de programa\Java\jre6\bin\javaw.exe (19/09/2011 03:43:48)
19
20
21
22
23
```

IF:

Només executa la porció de codi que hi ha en ella si la condició que li posem és certa.

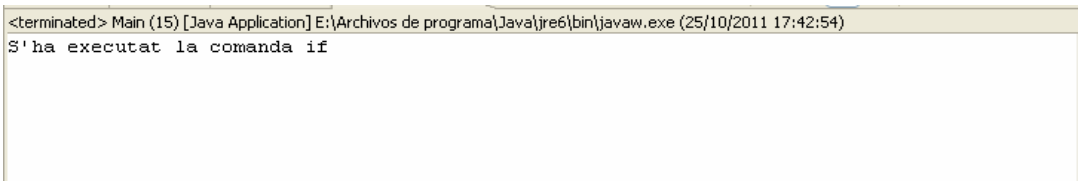
ELSE IF:

S'utilitza una vegada utilitzat el if i també s'executa quan la condició que hi té és certa però amb la diferència de que s'executarà només el primer que tingui la condició certa:

Exemple:

```
public class Main {

    public static void main(String[] args) {

        int x=14;

        if(x==14 && x<18){
            System.out.println("S'ha executat la comanda if");
        }
        else if(x==14 && x>11){
            System.out.println("S'ha executat la comanda else if");
            }
    }

}
```

Resultat:



FOR:

Estructura que es subdivideix en 3 trossos. El primer tros indica des de quan s'ha de començar la repetició, la segona indica fins on es repeteix i la tercera indica com es repeteix.

```
public class Main {

    public static void main(String[] args) {


        for(int i=1;i<6;i++){ // i++ es la abreviatura de i=i+1
            System.out.println(i+"a repetició");
        }
    }

}
```

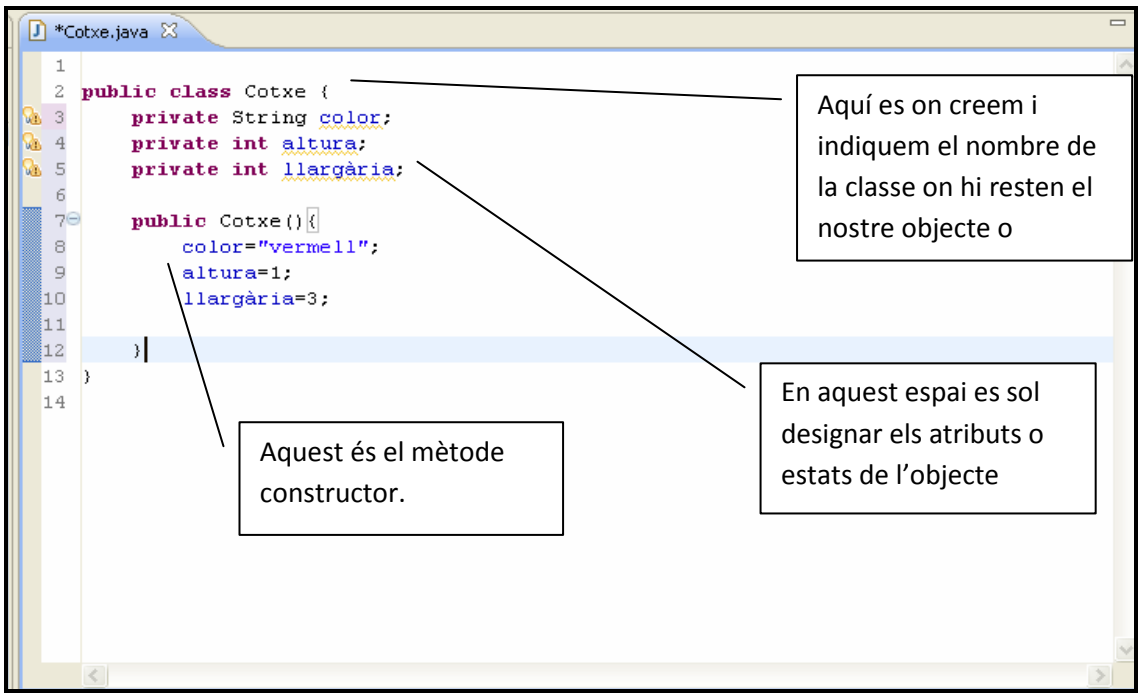
Aquesta comanda es llegiria de la següent manera: " Des de que la variable i és igual a 1 fins que la variable i sigui menor que 6 l'estructura es repeteixi d'un en un.

Resultat:

4.1.4 PROGRAMACIÓ ORIENTADA A OBJECTES

Una de les coses que fa especial a Java és el seu llenguatge orientat a objectes. Al nombrar objectes ens referim a qualsevol cosa que tingui un estat, un comportament i una identitat. L'estat d'aquest objecte és un atribut o una variable d'aquest objecte, és a dir una característica de l'objecte que sempre hi és, el comportament és el que pot fer aquest objecte i la identitat és el seu nombre. Totes

aquestes característiques estaran agrupades en classes, per tant una classe és un conjunt de codis on hi estan redactats els objectes i els seus comportaments. Prenem com a exemple un cotxe, l'estat d'aquest podria ser el seu color, la seva mida i el seu pes. El comportament serien accelerar, frenar ,girar cap a la dreta i/o girar cap a l'esquerra . I finalment la seva identitat seria "cotxe".



Davant del nombre de la classe hi ha una clau. Aquesta clau s'escriu al començament i final de cada classe i serveix per a la seva identificació. Aquest conveni també s'utilitza amb els mètodes. El mètode constructor descriu el comportament de l'objecte. A l'hora de crear els atributs s'han d'establir els seus permisos que es classifiquen en: "private", els atributs només es poden utilitzar dintre de la mateixa classe, "public", els atributs es poden utilitzar i modificar tant en la mateixa classe com un altre i "protected" (els atributs es poden llegir i modificar des de la mateixa classe, però no des de classes alienes o subclasses).

La classe que s'ha creat en la imatge només assigna els valors dels atributs, i per a posar-la en marxa haurem de crear un mètode main al qual haurem "d'instanciar" la classe. El que fem al instanciar es reservar una porció de memòria per a la classe que es llegirà quan iniciem el programa.

4.1.5 MÈTODES

Els mètodes són petites porcions de codi que realitzen una acció de l'objecte. Aquests es diferencien en:

- **Mètode constructor:** Igual que en l'apartat anterior aquest mètode ens indica les accions que farà la classe redactada. És el principal mètode d'una classe i s'invoca cada vegada que la classe és instanciada.
- **Mètode main:** És el primer mètode que s'identifica a l'hora de compilar ja que indica d'inicialització de l'execució del programa.
- **Mètode void:** Aquest mètode es caracteritza per no retornar ningun tipus d'objecte al fer les operacions demanades.
- **Mètodes que retornen un tipus d'objecte**.

4.1.6 LLIBRERIES

En programació es creen mètodes que s'utilitzen constantment i per això és útil crear llibreries per poder agrupar-les, guardar-les i utilitzar-les. Aquestes llibreries solen estar integrades en els entorns de desenvolupament o comprimides en arxius a part que s'han d'adherir al programa a desenvolupar. Les llibreries estan organitzades en forma ramificada, és a dir, les classes estan dintre d'unes altres que les agrupen i les classifiquen. Per poder fer-ne ús d'aquests trossos de codi només fa falta indicar al començament del programa quina llibreria utilitzarem (fent us de la sentencia *import*) indicant el nom de la llibreria i quina classe utilitzarem. Si utilitzem més d'una classe podem adherir tota la llibreria alhora per a estalviar-nos escriure la ruta de cada classe d'un en un, però és més recomanable aquesta última opció perquè estalvia a l'ordinador processar codi inutilitzat.

```
4  import java.awt.event.ActionListener;
5  import java.awt.image.BufferedImage;
6  import java.io.File;
7  import java.io.FileNotFoundException;
8  import java.io.FileOutputStream;
9  import java.io.IOException;
10
11 import javax.imageio.ImageIO;
12 import javax.swing.JButton;
13 import javax.swing.JCheckBox;
14 import javax.swing.JComboBox;
15 import javax.swing.JFileChooser;
16 import javax.swing.JFrame;
17 import javax.swing.JMenu;
18 import javax.swing.JMenuBar;
19 import javax.swing.JMenuItem;
20 import javax.swing.JScrollPane;
21 import javax.swing.JTextArea;
22 import javax.swing.JTextField;
23
24 import com.google.zxing.BarcodeFormat;
```

Per indicar dintre de quin directori resta la classe que importem ho fem amb un punt, que seria l'equivalent a la contrabarra en el sistema operatiu WINDOWS a l'indicar-li al navegador a quin arxiu del sistema es vol accedir. Els punts serveixen per a indicar que el membre de la dreta pertany o està dintre del membre de l'esquerra. Per tant a l'escriure la comanda *System.Out.Println()* indiquem que el mètode *Println()* està dintre de la subclasse *Out* de la classe *System*.

4.1.7 LA CLASSE JFRAME

Com anteriorment s'ha esmentat els entorns de desenvolupament compten amb diverses llibreries a les quals es pot accedir en qualsevol moment. En aquest apartat parlarem d'una classe fonamental en aquest ambient de programació, la classe JFrame. La utilitat bàsica d'aquesta classe és crear finestres en les quals s'afegiran components que formaran part de la part gràfica que interactuarà amb l'usuari. Per tant el nostre programa començarà important aquesta classe de la llibreria swing del paquet javax.swing.

Main.java | *exempleJFrame.java

```
1 import javax.swing.JFrame;
2
```

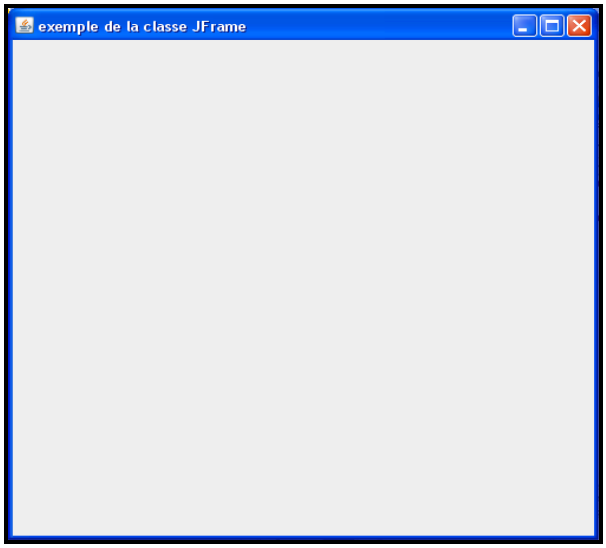
Seguidament s'indica a la classe que hereti els atributs de la classe JFrame. A l'hora d'heretar els atributs indiquem al programa que els utilitzarem des de la classe JFrame i per tant els podrem llegir i modificar. Després de realitzar aquests passos crearem la finestra en si donant-li un nom amb la comanda *super()* que invocarà la

classe JFrame de la superclasse JComponent escrivint entre els parèntesis el nom en format string de la finestra. Els parèntesis de les comandes serveixen per a introduir característiques per l'usuari que complementaran la funció de la comanda, és per això que totes les comandes, mètodes, classes conten amb aquests parèntesi. Una vegada creada la finestra li assignem una mida, li indiquem com ha d' ordenar el components que afegirem a la finestra i com s'ha de tancar. Ara tindrem la finestra creada però serà invisible, i per tant en l'últim pas haurem d'indicar si volem que la finestra resti invisible o adquireixi visibilitat.

Main.java *exempleJFrame.java

```
1  import javax.swing.JFrame;
2
3
4  public class exempleJFrame extends JFrame{
5
6      public exempleJFrame(){
7          super("exemple de la classe JFrame");
8          setSize(500,500); //MIDA DE LA FINESTRA
9          setLayout(null); //ELS COMPONENTS ADHERITS NO S'ORGANITZARAN
10         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);// COM ES TANCARÀ LA FINESTRA
11         setVisible(true); // S'INDICA SI LA FINESTRA ES VISIBLE O INVISIBLE
12     }
13
14
15 }
16
```

El resultat :



4.1.8 LA CLASSE JPANEL

Aquesta classe és utilitzada com a llenç per a representar-hi figures. Com que no consta de finestra pròpia la complementarem amb la classe *JFrame* per poder visualitzar-la. El mètode que s'encarrega de la representació de les figures és *paintComponent(),* que pertany a la classe *JComponent* de la llibreria swing. Per a l' utilització de *JPanel* es comença important la classe del paquet *java.swing* .Seguidament s'hereta la classe utilitzant la comanda *extends* i en el mètode constructor nomes fa falta posar la mida del llenç. Per a dibuixar es crea un mètode que tingui el nom paint i se'l passa com a argument la classe *Graphics*. Aquesta classe serà la creadora de figures geomètriques i de donar-li color. Sintetitzant, el que s'aconsegueix en aquest procés és crear una figura amb la classe *Graphics* i representar-la amb *paintComponent()* en un suport donat per la classe *JPanel.* Però com abans s'ha esmentat, per poder visualitzar aquests components necessitarem una finestra visible, per tant creant una classe que hereti els mètodes de JFrame i instanciant el llenç i afegint-lo a la classe de la finestra amb la comanda *add()* resolem el problema. El programa ha de quedar semblant a aquest:

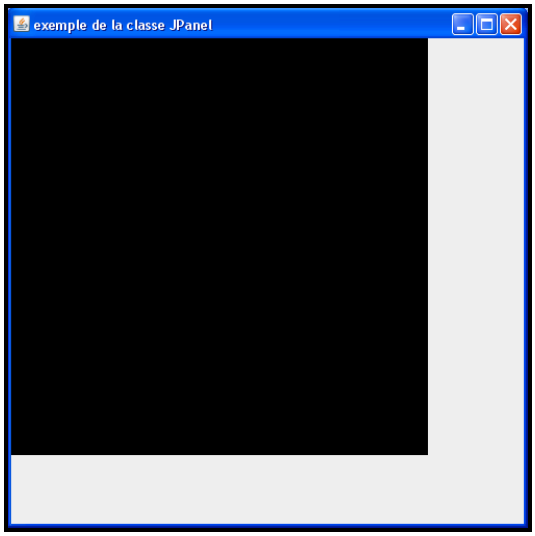
exempleJPanel.java | Main.java | Finestra.java

```
1  import java.awt.Color;
2  import java.awt.Graphics;
3  import javax.swing.JPanel;
4
5
6  public class exempleJPanel extends JPanel{
7
8      public exempleJPanel(){
9          this.setSize(400,400);
10     }
11
12     public void paint(Graphics g){
13         g.setColor(Color.BLACK);
14         g.fillRect(0, 0, 400, 400);
15
16     }
17 }
```

exempleJPanel.java | Main.java | *Finestra.java

```
1  import javax.swing.JFrame;
2
3
4  public class Finestra extends JFrame{
5
6  private exempleJPanel jp;
7
8      public Finestra(){
9          super("exemple de la classe JPanel");
10         setSize(500,500);
11         setLayout(null);
12         jp=new exempleJPanel();//INSTANCIA DEL LLENÇ
13         add(jp); //AFEGIM EL LLENÇ A LA FINESTRA
14         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15         setVisible(true);
16     }
17 }
```

I el resultat :



La classe Graphics compta amb mètodes que creen polígons, ovals, rectangles, punts, línees e imatges. Tots aquest es representen en un eix cartesià imaginari que té com a origen la cantonada esquerra superior del llenç. Horitzontalment cap a la dreta estan representats els nombres (x,0) de l'eix i verticalment cap avall hi son els nombres (0,y) en els quals cada nombre representa un píxel de la pantalla. A més de la posició s'ha d'indicar la mida i el color, aquest últim s'indica al començament del mètode de la figura a definir i prové d'una classe aliena a la classe *Graphics* que es la classe *Color* de la llibreria *awt*.

4.1.9 TRACTAMENT D'IMATGES

En un sistema operatiu només faria falta clicar damunt la imatge desitjada i "voilà", ja tenim representada la imatge, però en programació no és així. Hem de tenir en compte moltes coses, ja sigui la memòria que se li assignarà, la mida de la imatge, on es representa, quan es representa i com es representa. Per sort comptem amb la classe *bufferedImage* que conté diversos mètodes relacionats amb el tractament d'imatges,d' un dels quals parlarem en aquest apartat. Una de les utilitats remarcables d'aquesta classe és la tècnica de doble buffer que utilitza per representar les imatges, és a dir, primer carrega la imatge en un buffer (espai de memòria on s'emmagatzema informació durant una transferència) fora de la pantalla i després la carrega en la pantalla, aquesta tècnica que és realment útil en animacions resol problemes com la obtenció d'una imatge a trossos per algun

error. A més obliga a manegar objectes *bufferedImage* que donen la possibilitat de modificar i estudiar la imatge amb altres mètodes . Aquesta classe pertany al paquet *Image* de la llibreria *awt.* Per poder dibuixar una imatge en un llenç com el que s'ha programat anteriorment és imprescindible crear un mètode que carregui la imatge en el buffer. Aquest procés pot tornar errors i és per aquesta raó que s'escriu el codi per carregar la imatge dintre de les comandes *try* i *catch*. Aquestes comandes estableixen un tipus de filtre que detecta els errors i els classifica per mostrar-nos en pantalla de quin tipus d'error es tracta. Per carregar la imatge en el buffer farà falta importar la classe *imageIO* que consta d'un mètode, concretament el mètode read(), que s'encarrega de tot el procès de lectura. És important aclarir que aquest mètode necessita com argument un objecte del tipus File, és a dir, la ruta de la imatge. Una vegada carregada la imatge indiquem al programa que ens la representi utilitzant la comanda *drawImage* de la classe *Graphics* anteriorment descrita.

El programa ha de quedar semblant a aquest:

*exempleJPanel.java | Finestra.java | Main.java

```
1  import java.awt.Color;
2  import java.awt.Graphics;
3  import java.awt.image.BufferedImage;
4  import java.io.File;
5  import javax.imageio.ImageIO;
6  import javax.swing.JPanel;
7
8
9  public class exempleJPanel extends JPanel{
10     BufferedImage imatge;
11
12     public exempleJPanel(){
13         this.setSize(400,400);
14         cargarImatge();// EN EL METODE CONSTRUCTOR INDIQUEM
15     }//QUE S'INICIALITZI EL MÈTODE cargarImatge()
16
17     public void paint(Graphics g){
18         g.setColor(Color.BLACK);
19         g.drawImage(imatge, 0, 0,null);
20     }
21     public void cargarImatge(){
22         try{
23             File rutaImatge=new File("RUTA DE LA IMATGE");
24             imatge=ImageIO.read(rutaImatge);
25         }catch(Exception e){
26             e.printStackTrace();
27         }
28     }
29 }
```

Resultat:

5. COM ES PROGRAMA UN PROGRAMA CODIFICADOR DE CODIS QR

L'objectiu d'aquest programa es convertir un conjunt de lletres i/o números en un codi QR. Aquest codi QR s'ha de representar visualment dintre del programa i ha d'oferir la possibilitat d'exportar el codi en format imatge a qualsevol carpeta del sistema. A més hem de poder elegir la mida del codi i el nivell de correcció d'errors alhora de codificar la informació.

5.1 CODIFICACIÓ D'UN CODI QR AMB LA LLIBRERIA ZXING

La llibreria ZXING (pronunciada zebra crossing) conté diverses classes pel tractament de codis de barres UPC-A, UPC-E, EAN-8, EAN-13, CPDE 128, QR CODE, ITF, CODABAR, RSS-14, DATA MATRIX, PDF 147 i AZTEC. Amb aquesta llibreria podem analitzar imatges,identificar el codi en qüestió i descodificar-ho, a més conté el material suficient per la codificació d'un codi QR. Les llibreries estan disponibles en llenguatge JAVA, JAVA per Android i per iPhone.

La codificació d'un codi QR podem dividir-la en 8 passos:

1. **Anàlisis de la informació:**

Primer de tot s'analitza la informació a codificar i s'estableix el mode de codificació per estalviar el màxim de bits per fer-ho. Es poden utilitzar més d'un mode a la vegada però per la complexitat de la operació, és més senzill utilitzar-ne només un per a tot el codi. El mètode de la llibreria ZXING encarregat de fer aquesta operació és chooseMode () de la classe Mode(). En la següent imatge podem veure un tros de codi on es realitza aquesta classificació.

```
public static Mode chooseMode(String content, String encoding) {
  if ("Shift_JIS".equals(encoding)) {
    // Choose Kanji mode if all input are double-byte characters
    return isOnlyDoubleByteKanji(content) ? Mode.KANJI : Mode.BYTE;
  }
  boolean hasNumeric = false;
  boolean hasAlphanumeric = false;
  for (int i = 0; i < content.length(); ++i) {
    char c = content.charAt(i);
    if (c >= '0' && c <= '9') {
      hasNumeric = true;
    } else if (getAlphanumericCode(c) != -1) {
      hasAlphanumeric = true;
    } else {
      return Mode.BYTE;
    }
  }
  if (hasAlphanumeric) {
    return Mode.ALPHANUMERIC;
  } else if (hasNumeric) {
    return Mode.NUMERIC;
  }
  return Mode.BYTE;
}
```

2. **Codificació de la informació:**

En aquesta part la informació es converteix en un flux de bits segons el mode utilitzat. Després s'introdueix un indicador de mode davant de cada subconjunt de dades per saber en quin mode estan codificats. Finalment s'introdueix un terminador i es divideixen les dades en grups de 8 bits (codewords).

```
BitArray dataBits = new BitArray();
appendBytes(content, mode, dataBits, encoding);
```

3. **Crear espai en la memòria**

Reservem espai en la memòria per afegir els bits que s'han codificat.

```
int numInputBytes = dataBits.getSizeInBytes();
initQRCode(numInputBytes, ecLevel, mode, qrCode);
```

4. **Afegir la informació de la versió i el format**

Es crea un vector de bits amb el numero de la versió i el format i s'afegeixen al flux de bits creats amb la informació.

```
BitArray headerAndDataBits = new BitArray();
```

5. Codificació de la correcció d'errors

Per a què el codi disposi de màxima seguretat envers agents externs que puguin alterar la seva informació, s'executa un algoritme de correcció d'errors per generar codewords de correcció d'errors que es disposaran al final dels codewords d'informació. A més en aquesta part s'afegeix un segment de codi per a omplir la matriu si fa falta

```
BitArray finalBits = new BitArray();
interleaveWithECBytes(headerAndDataBits, qrCode.getNumTotalBytes(), qrCode.getNumDataBytes(),
    qrCode.getNumRSBlocks(), finalBits);
```

```
terminateBits(qrCode.getNumDataBytes(), headerAndDataBits);
```

6. Emmascarar les dades

Establir un patró de màscara basat en aritmètica modular seguint la una de les fórmules descrita en la taula de l'apartat "CODI QR".

```
ByteMatrix matrix = new ByteMatrix(qrCode.getMatrixWidth(), qrCode.getMatrixWidth());
qrCode.setMaskPattern(chooseMaskPattern(finalBits, qrCode.getECLevel(), qrCode.getVersion(),
    matrix));
```

7. Fer la matriu

En aquest pas es fa la matriu amb els bits codificats i finalment es disposen les dades de tal manera que formin un codi QR.

```
MatrixUtil.buildMatrix(finalBits, qrCode.getECLevel(), qrCode.getVersion(),
    qrCode.getMaskPattern(), matrix);
```

8. Verificació

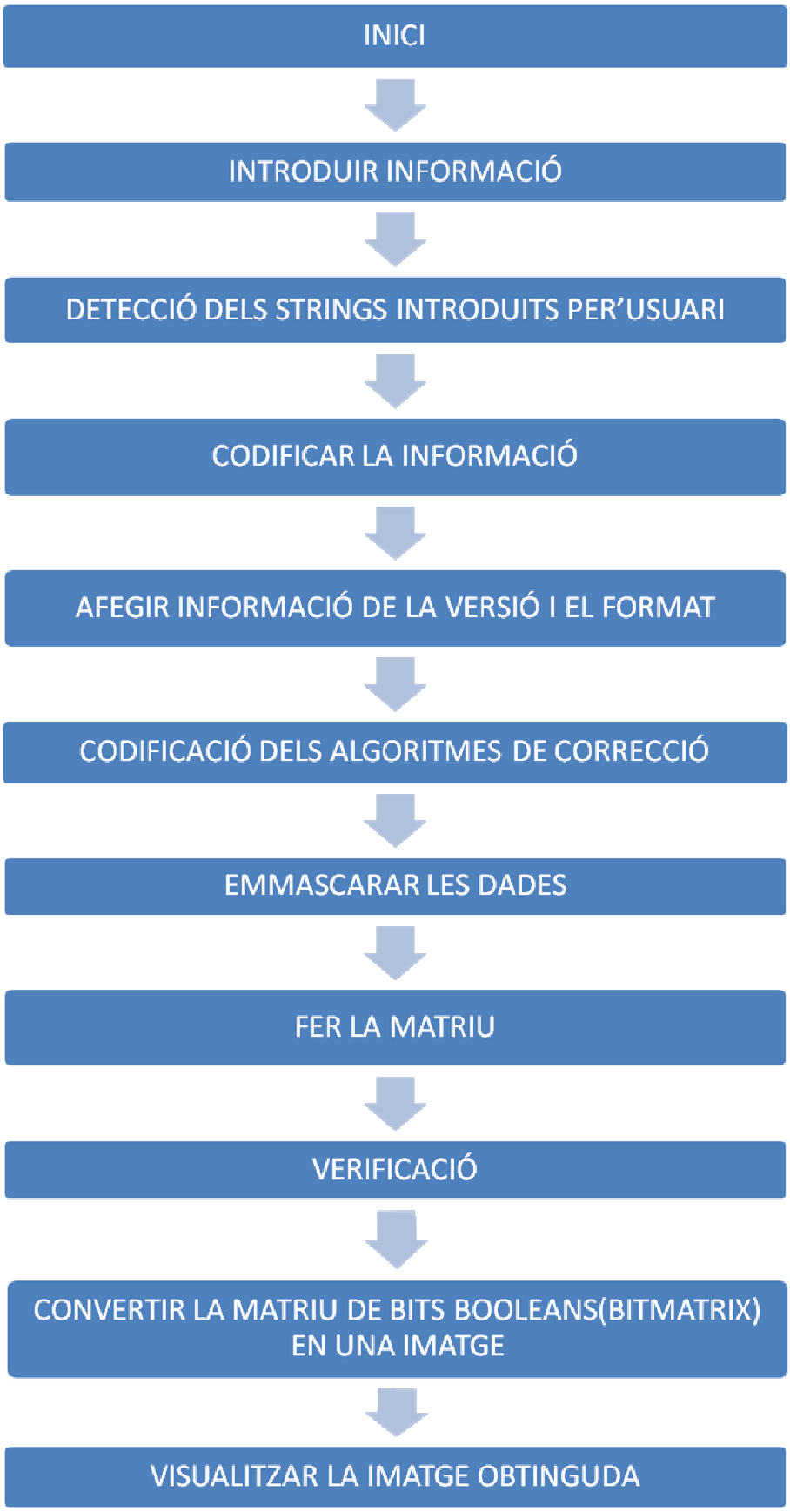
Per a assegurar que les dades introduïdes poden formar un codi QR, es verifica si realment s'ha creat un codi QR.

```
if (!qrCode.isValid()) {
  throw new WriterException("Invalid QR code: " + qrCode.toString());
```

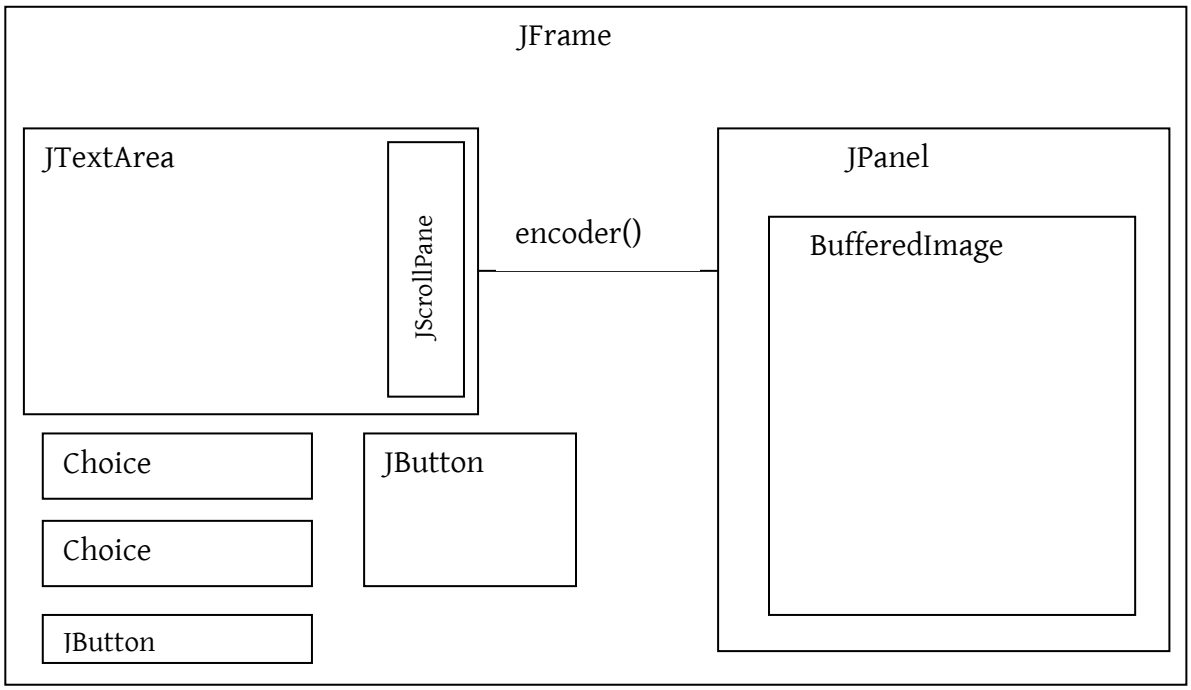
Amb aquests mètodes podríem crear un codi a partir d'un flux de Strings (lletres) però a més de complir l'objectiu de crear un codi QR ha de comptar amb suficients elements visuals per a poder introduir una certa informació i convertir-la en un codi QR sense alterar el codi del programa. Per aquesta raó es farà ús de les classes JFrame, JPanel, JText i JButton per poder crear un programa que es comuniqui visualment amb l'usuari.

Al fer un programa podem començar programant des de la part més externa del programa(els elements visuals que es comunicaran amb l'usuari) fins la part més interna (la part de codi que dóna sentit al programa i designa les funcions d'aquests elements visuals) o a l'inrevés, personalment prefereixo programar la part més interna primer, perquè ens estalviarem escriure tros de codi si per alguna raó el programa falla.

5.2 DIAGRAMA DE CODIFICACIÓ



5.3 ESQUEMA VISUAL DE LES CLASSES QUE S'UTILITZARAN EN EL PROGRAMA



5.4 PROGRAMA CODIFICADOR DE CODIS QR:

Començarem programant el mètode que transformi un flux de strings en un codi QR. Per a aquest procés necessitarem la llibreria *ZXING* que contindrà les classes necessàries per a fer-ho. Per aquesta raó crearem una classe en el nostre entorn de desenvolupament i importarem les subclasses *BitMatrix., ErrorCorrectionLevel, ByteMatrix, Encoder i QRCode.* També introduïm com a atributs un objecte del tipus *BufferedImage* que serà la imatge del codi QR i un objecte del tipus BitMatrix que és un objecte fet d'una matriu booleana. En aquesta classe creem un mètode amb el nom de *codificar* i li introduïm com paràmetres un String i dos ints. Aquest String serà la informació que introduirem a codificar i els dos ints seran les mesures del codi a crear. Seguidament instanciem la classe *QRCode* que posteriorment convertirà el flux de Strings en una matriu de bits que formaran un objecte *QR CODE* (un codi qr), aprofitant la subclasse *encode* que esta en la classe *Encoder* de la llibreria *ZXING* li donem forma al objecte QR CODE entre les comandes *try* i *catch,* explicades degudament en l'apartat *BufferedImage,*. Ens demanarà com a atributs del mètode un objecte del tipus *String*, és la informació a codificar, un

valor que indiqui la seguretat del codi (vegeu apartat "EL CODI QR" per més informació) , i un objecte tipus *QR CODE* que serà el codi codificat en si. A l'hora de demanar-nos quina és la seguretat del codi posem "H", però més endavant haurem de modificar-ho per a que puguem elegir qualsevol mode a l'hora de crear-ho.

```
public void codificador(String informacio,int amplada, int alçada){
QRCode qr=new QRCode();
imatgeCodificada=new BufferedImage(amplada,alçada,BufferedImage.TYPE_4BYTE_ABGR);
try {
    Encoder.encode(informacio, ErrorCorrectionLevel.H, qr);
} catch (Exception e) {
    e.printStackTrace();
}
}
```

Ara només falta mostrar el resultat en pantalla. Desgraciadament no ho podem fer directament, necessitem passar aquest objecte del tipus QR CODE a un objecte BitMatrix per poder manejar-ho de tal manera que puguem crear una imatge a partir d'aquest i representar aquesta imatge en un llenç. Per crear una matriu de bits boolean hem de crear un mètode que transformi el QR CODE en un objecte del tipus BitMatrix d'aquesta manera podrem variar la seva mida alhora. Anomenem a aquest mètode *ampliarMida()* i li introduïm com a paràmetres un codi QR , i dos ints que representaran la mida de la matriu de bits booleans resultant. És important aclarir que encara que puguem triar entre diferents mesures d'alçada i d'amplada el codi resultant sempre tindrà forma quadrada, l'únic que variaran seran els bits de farcit que es troben al voltant seu. Comencem el mètode instanciant l'objecte del tipus CODE QR com a una matriu de bytes (bytes i bits són coses diferents) i les mides del codi d'entrada, després instanciem les mides del codi d'entrada més els bits de farcit. Una vegada instanciades aquestes mides, creem les mides de sortida, que hand de ser més grans que les de entrada, i per assegurar-nos que és així utilitzem la comanda *Math.max(int a, int b).* Després busquem un múltiple entre les mides del codi QR de sortida i el d'entrada, que utilitzarem posteriorment. I com a últim pas calculem la mida del farcit del codi QR de sortida. El procés que explica com es fan els càlculs esmentats està explicat en les imatges següents.

```
ByteMatrix input = codiQR.getMatrix();
int ampladaEntrada = input.getWidth();
int alçadaEntrada = input.getHeight();
int qrAmplada = ampladaEntrada + (4 << 1);//(4<<1) fa referencia als bits que
int qrAlçada = alçadaEntrada + (4 << 1);//es deixen en blanc al voltant del codi
int ampladaSortida = Math.max(amplada, qrAmplada);
int alçadaSortida = Math.max(alçada, qrAlçada);

int multiple = Math.min(ampladaSortida / qrAmplada, alçadaSortida / qrAlçada);
int farcitEsquerre= (ampladaSortida - (ampladaEntrada * multiple)) / 2;
int farcitSuperior = (alçadaSortida - (alçadaEntrada * multiple)) / 2;
```

Totes aquestes mesures les utilitzarem per omplir els bits d'una matriu que tindrà les dimensions que nosaltres hem demanat a l'instanciar el mètode *ampliarMida(QRCode codiQR,* ***int amplada, int alçada****)* en la que cada bit serà de la mesura del int multiple que acabem de crear. Per tant instanciem la matriu de bits de sortida i l'omplim seguint la forma indicada en la imatge.

```
BitMatrix sortida = new BitMatrix(ampladaSortida, alçadaSortida);

for (int Yentrada = 0, outputY = farcitSuperior; Yentrada < alçadaEntrada; Yentrada++, outputY += multiple) {
    for (int Xentrada = 0, outputX = farcitEsquerre; Xentrada < ampladaEntrada; Xentrada++, outputX += multiple) {
        if (input.get(Xentrada, Yentrada) == 1) {
            sortida.setRegion(outputX, outputY, multiple, multiple);
    }
  }
}

return sortida;
}
```

El que hem fet en la imatge anterior és indicar a l'ordinador que comenci a omplir els bits de la matriu resultant des que acaba el farcit i deixi d'omplir-los quan la matriu d'entrada ja no li en queden. Com els valors booleans no tenen mida indiquem que els vagi introduint segons en una regió predeterminada per la mida *multiple* que es va instanciar anteriorment.

Després instanciem la imatge a la qual havíem creat l'atribut del tipus *BufferedImage* perquè aprofitem el mateix mètode que converteix els Strings en una matriu booleana per transformar la matriu booleana a una matriu de llindars de colors blanc i negre en un format d'imatge. Una vegada obtenim la matriu booleana hem de transformar-la en una imatge. La imatge ja la tenim instanciada però hem d'introduir-li la informació que la formarà, és a dir, hem d'assignar-li un color a cada un dels píxels que la forma. Per fer-ho utilitzarem l'operador lògic *for* dues vegades (un dintre d'un altre) que indicarà a l'ordinador que recorri la matriu d'esquerra a dreta i de dalt a baix i que segons trobi un true o un false en la matriu boleana implementi un píxel blanc o negre en la imatge.

```
for(int y=0;y<bm.height;y++){
    for (int x=0;x<bm.width;x++){
        if(bm.get(x, y)){
            imatgeCodificada.setRGB(x, y, -16777216);
        }
        else
            imatgeCodificada.setRGB(x, y, -1);
    }
}
```

Com podem veure en la imatge no hem trobat la necessitat d'igualar la sentencia a què hem posat dintre de l'if a *true* perquè la sentència en si ha de ser o vertadera o falsa ja que la matriu és de tipus booleana. La comanda *setRGB()* ens demana una posició i un número int. Aquest numero *int* ha d'equivaler a un color que convertit en bytes ha de contenir les quantitats de vermell, verd i blau que s'han de barrejar a més d'un altre byte que ha de contenir el nivell de transparència del color, en aquest cas l'int -16777216 equival al negre i el -1 al blanc.

Una vegada programat el creador de codis QR, implementem els elements esmentats en l'esquema del programa. Començarem implementant la classe JFrame en la mateixa classe que s'ha creat el mètode codificador i introduirem els mateixos paràmetres que en l'exemple anterior a diferència de la mida, que en aquest cas serà 800 X 560 píxels i introduirem el mètode *setResizable(false)* per a que la finestra resti estàtica i no canviï la seva mida (evitem possibles problemes que puguin donar el reposicionament de la finestra).

```
public class codificar extends JFrame{
    private BitMatrix bm ;
    private BufferedImage imatgeCodificada;

    public codificar(){
        super("CODIFICADOR DE CODIS QR");
        setSize(800,600);
        setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setResizable(false);
        setVisible(true);
    }
```

Paral·lelament crearem una classe que implementi la classe JPanel que mostrarà la imatge del codi QR a l'usuari i que contindrà la classe *codificar()* com a paràmetre. Li donem com a paràmetre la classe de la finestra per a què puguem utilitzar el mètode *codificador()* que acabem de crear. Després d'assignar-li una mida al llenç

creem el mètode *paintComponent(Graphics g)* que s'encarregarà de representar el codi QR. Per poder diferenciar el panell de la resta de la finestra, primer de tot, dibuixem un quadrat blanc i després un quadrat buit de color negre per a diferenciar el llenç de la resta en la finestra, posteriorment, indiquem dintre d'una sentencia *if()* si la imatge no està buida, que la representi. Finalment introduïm la comanda *repaint()* per a què refresqui el llenç i així aparegui el codi QR una vegada s'hagi creat. S'entendrà millor en la imatge:

```
public class Llenç extends JPanel{
    private codificar c;

    public Llenç(codificar c){
        this.c=c;
        this.setSize(501,501);
    }

    public void paintComponent(Graphics g){
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, 501, 501);
        g.setColor(Color.BLACK);
        g.drawRect(0,0,500,500);
        if(c.getimatgeCodificada()!=null)
            g.drawImage(c.getimatgeCodificada(),0,0,null);
        repaint();
    }
}
```

Per comprovar que el mètode constructor esta ben programat fem una prova: Instanciem el llenç en la finestra, indiquem la posició del llenç en la finestra amb la comanda *setBonds()* i instanciem el mètode *codificador()* amb els paràmetres a l'atzar. Si està ben programat el panell ha de mostrar la imatge del codi, si no, la consola ens dirà quin és l'error i on es troba.

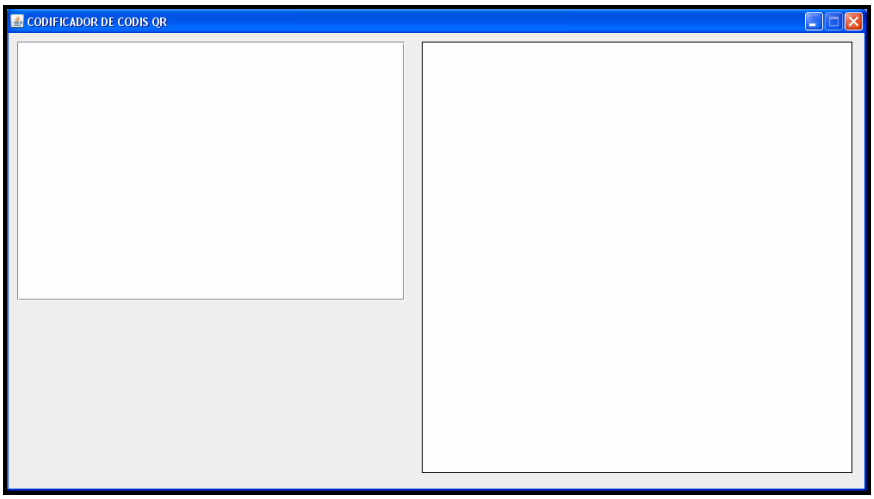
Les dues classes han de semblar-se a aquestes imatges per a obtenir el següent resultat:

```
public class Llenç extends JPanel{
    private codificar c;

    public Llenç(codificar c){
        this.c=c;
        this.setSize(501,501);
    }

    public void paintComponent(Graphics g){
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, 501, 501);
        g.setColor(Color.BLACK);
        g.drawRect(0,0,500,500);
        if(c.getimatgeCodificada()!=null)
            g.drawImage(c.getimatgeCodificada(),0,0,null);
        repaint();
    }
}
```

```
public class codificar extends JFrame{
    private BitMatrix bm ;
    private BufferedImage imatgeCodificada;

    public codificar(){
        super("CODIFICADOR DE CODIS QR");
        setSize(800,560);
        setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setResizable(false);
        setVisible(true);
        Llenç l = new Llenç(this);
        add(l);
        codificador("LA",500,500);
    }
```

Resultat:

Ja ens hem assegurat que el programa codifica be, però encara ens falten per programar les classes que permetran introduir un text qualsevol al programa per a què el codifiqui. Ara és quan intervenen les classes *JTextArea* i *JScrollPane*. *JTextArea* és la porció de la finestra on introduirem la informació a codificar i *JSCrollPane* és un eina que permetrà desplaçar-nos pel text que s'hagi introduït si aquest és més gran que la porció delimitada. Seguidament importem les dues en la classes a la finestra i les establim com a atributs del programa. Després les instanciem però s'ha de anar amb comte que a l'instanciar la classe *JSCrollPane* se li ha de introduir com a paràmetre l'objecte *JTextArea*. Amb aquesta operació haurem creat un objecte en el qual podrem escriure un text, i ens podrem desplaçar lliurement pel que hàgim escrit sense restriccions. Adherim aquest objecte a la finestra amb la comanda *add()* i establim la posició que ocuparà.



Él metode setLineWrap, establirà automàticament un salt de línea en l'objecte JFileChooser quan el text escrit sigui d'una mida major que la mateixa línea. Ja es l'hora de crear el botó que agafarà el text introduït en l'objecte JTextArea i començarà la seva codificació. Aquest és l'objecte *JButton* de la llibreria *javax* que haurem de introduir-lo en la classe de la finestra com a atribut i instanciar-lo en el mètode constructor d'aquesta. A l'instanciar-lo necessitarà un paràmetre String, que és el nom que tindrà el botó en la finestra, adherim el botó a la finestra i introduïm les mides i la posició que ocuparà. Una vegada instanciat el botó és hora d'implementar una interfície. Les interfícies son conjunts de mètodes que defineixen el que una classe pot fer, és a dir, estableixen una relació directe entre el món exterior i el codi del programa. Bé, implementem la interfície actionListener en la finestra per a que puguem donar-li una funció al botó que hem instanciat.

```
public class codificar extends JFrame implements ActionListener{
```

Una vegada implementada, fem que el botó filtri qualsevol acció que es faci sobre ell (bàsicament si es clicat).

```
boto.addActionListener(this);
```

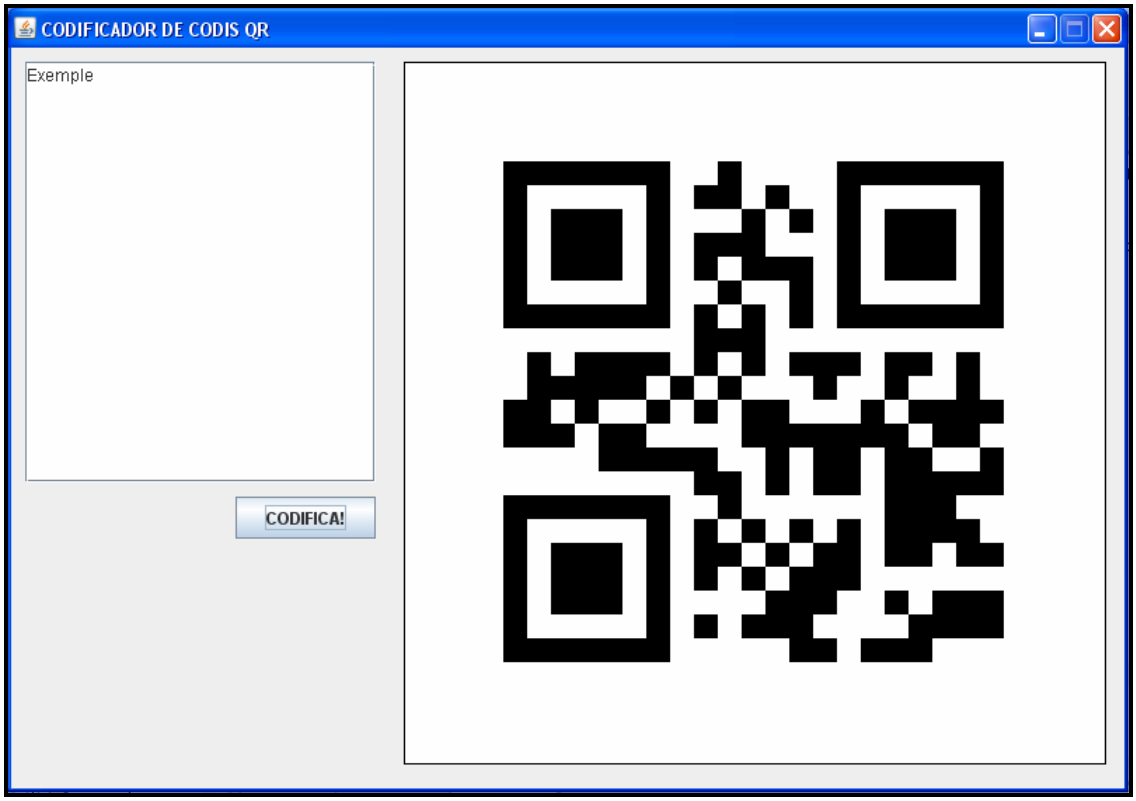
D'aquesta manera podrem saber quan el botó és clicat i així podrem dur a terme les accions necessàries per codificar el text introduït en l'objecte JTextArea. Implementem els mètodes de la interfície que s'encarregaran d'observar en aquest cas quan el boto és clicat. Dintre d'aquest mètode introduïm l'operador lògic if que farà la funció d'agafar el text del JTextArea i utilitzar-lo en el mètode codificador quan l'acció realitzada en la finestra sigui la de clicar el botó.

```
public void actionPerformed(ActionEvent e) {
    if(e.getActionCommand()==boto.getActionCommand()){
        String informacio=textArea.getText();

    }

}
```

Comprovem si la interfície funciona correctament i si hem utilitzat correctament els mètodes instanciant en aquest últim el mètode codificador amb la informació que li introduïm al JTextArea, i les mides a l'atzar.

```
public void actionPerformed(ActionEvent e) {
    if(e.getActionCommand()==boto.getActionCommand()){
        String informacio=textArea.getText();
        codificador(informacio,500,500);
    }

}
```

Al posar "Exemple" en el JTextArea i clicar el botó "Codifica!" aquest és el resultat que
obtenim:



Encara falten els botons que seleccionaran la mida i el nivell de correcció d'error en el programa. Aquest botons són objectes de tipus Choice que els crearem com a atributs juntament amb els seus respectius vectors de Strings. Aquests vectors de String contindran el text que es mostrarà en aquests Choice, que bàsicament són un tipus de menú en els quals podem elegir una opció. Instanciem el vector de String que contindrà els textos que mostraran les mides.

```
mides=new String[4];
mides[0]="Seleccioni mida";
mides[1]="300x300";
mides[2]="400x400";
mides[3]="500x500";
```

Aquestes mides estaran representades dintre del objecte Choice de la següent manera:

```
choice1=new Choice();
choice1.add(mides[0]);
choice1.add(mides[1]);
choice1.add(mides[2]);
choice1.add(mides[3]);
add(choice1);
choice1.setBounds(10,320,130,30);
```

Realitzem el mateix procés per poder triar el nivell de correcció d'errors :

```
nce=new String[5];
nce[0]="Seleccioni ECL";
nce[1]="H";
nce[2]="L";
nce[3]="M";
nce[4]="Q";
choice2=new Choice();
choice2.add(nce[0]);
choice2.add(nce[1]);
choice2.add(nce[2]);
choice2.add(nce[3]);
choice2.add(nce[4]);
add(choice2);
choice2.setBounds(10,360,130,30);
```

Només falta programar els mètodes que segons quins valors s'hagin seleccionat en aquests menus ens torni un valor int o ErrorCorrectionLevel i no en format String. Per això crearem un mètode del tipus int que retorni una mida depenent d'allò que s'hagi seleccionat en el menú.

```
public int mida(){
int mida=0;
    if(choice1.getSelectedIndex()==1){
        mida=300;
    }
    else if(choice1.getSelectedIndex()==2){
        mida=400;
    }
    else if(choice1.getSelectedIndex()==3){
        mida=500;
    }
    return mida;
}
```

Realitzem el mateix procés per a seleccionar el nivell de correccions d'errors

```
public ErrorCorrectionLevel ecl(){
    ErrorCorrectionLevel ecl=null;
    if(choice2.getSelectedIndex()==1){
        ecl=ErrorCorrectionLevel.L;
    }
    else if(choice2.getSelectedIndex()==2){
        ecl=ErrorCorrectionLevel.M;
    }
    else if(choice2.getSelectedIndex()==3){
        ecl=ErrorCorrectionLevel.Q;
    }
    else if(choice2.getSelectedIndex()==4){
        ecl=ErrorCorrectionLevel.H;
    }
    return ecl;
}
```

Com havíem esmentat al començament de la redacció del mètode codificador, arribaria un punt en el qual hauríem de canviar part del codi programat. S'ha de afegir un quart paràmetre al mètode *codificador()* a part del String d'informació i els dos ints de mesures. Aquest quart serà del tipus ErrorCorrectionLevel i per tant a l'instanciar la matriu QR podrem triar el tipus de nivell de correcció d'errors que vulguem. Per tant el mètode passarà de ser:

```
public void codificador(String informacio,int amplada, int alçada){
QRCode qr=new QRCode();
imatgeCodificada=new BufferedImage(amplada,alçada,BufferedImage.TYPE_4BYTE_ABGR);
try {
    Encoder.encode(informacio,ErrorCorrectionLevel.Q, qr);
    bm=ampliarMida(qr,amplada,alçada);
} catch (Exception e) {
    e.printStackTrace();
}
for(int y=0;y<bm.getHeight();y++){
    for(int x=0;x<bm.getWidth();x++){
        if(bm.get(x, y)){
            imatgeCodificada.setRGB(x, y, -16777216);
        }
        else{
            imatgeCodificada.setRGB(x,y,-1);
        }
    }
}
}
```

A ser:

```
public void codificador(String informacio, ErrorCorrectionLevel ecl,int amplada, int alçada){
QRCode qr=new QRCode();
imatgeCodificada=new BufferedImage(amplada,alçada,BufferedImage.TYPE_4BYTE_ABGR);
try {
    Encoder.encode(informacio,ecl, qr);
    bm=ampliarMida(qr,amplada,alçada);
} catch (Exception e) {
    e.printStackTrace();
}
for(int y=0;y<bm.getHeight();y++){
    for(int x=0;x<bm.getWidth();x++){
        if(bm.get(x, y)){
            imatgeCodificada.setRGB(x, y, -16777216);
        }
        else{
            imatgeCodificada.setRGB(x,y,-1);
        }
    }
}
```

Una vegada acabat el programa busquem la manera d'exportar la imatge a l'exterior per a què l'usuari pugui utilitzar-la. Per aquesta raó inclourem un botó extra al programa que s'encarregarà de treure la imatge del programa i traslladar-la en una carpeta designada per l'usuari. És a dir, crearem un objecte que permeti triar a l'usuari en quina carpeta vol desar la imatge. Per sort no tindrem la necessitat de "crear-ho" ja que la llibreria de java en té un, concretament amb JFileChooser de la llibreria javax.Swing . Indicarem al programa que al clicar el botó de desar, automàticament s'obri l'objecte JFileChooser i guardi la imatge. Per fer-ho crearem un int que contingui la variable JFileChooser.ShowSaveDialog(this) dintre d'un if que contindrà com a condició prémer la tecla de desar. Al crear aquesta variable automàticament s'obrirà la finestra per a desar l'objecte i és quan

es tancarà, que aquesta variable tindrà un valor int. Dintre d'un altre if posem com a condició que aquesta variable sigui igual al nombre int que ens retorna la finestra al clicar el botó guardar del JFileChooser. D'aquesta manera s'engegarà la funció que escrivim dintre d'aquest if al prémer el botó desar de la finestra de guardar. Seguidament creem un métode try i catch i dintre del try a la comanda ImageIO.write() li introduïm com a paràmetres la imatge a guardar, el format i on la ruta on es guardarà, en aquest últim paràmetre indiquem la ruta seleccionada en el JFileChooser.

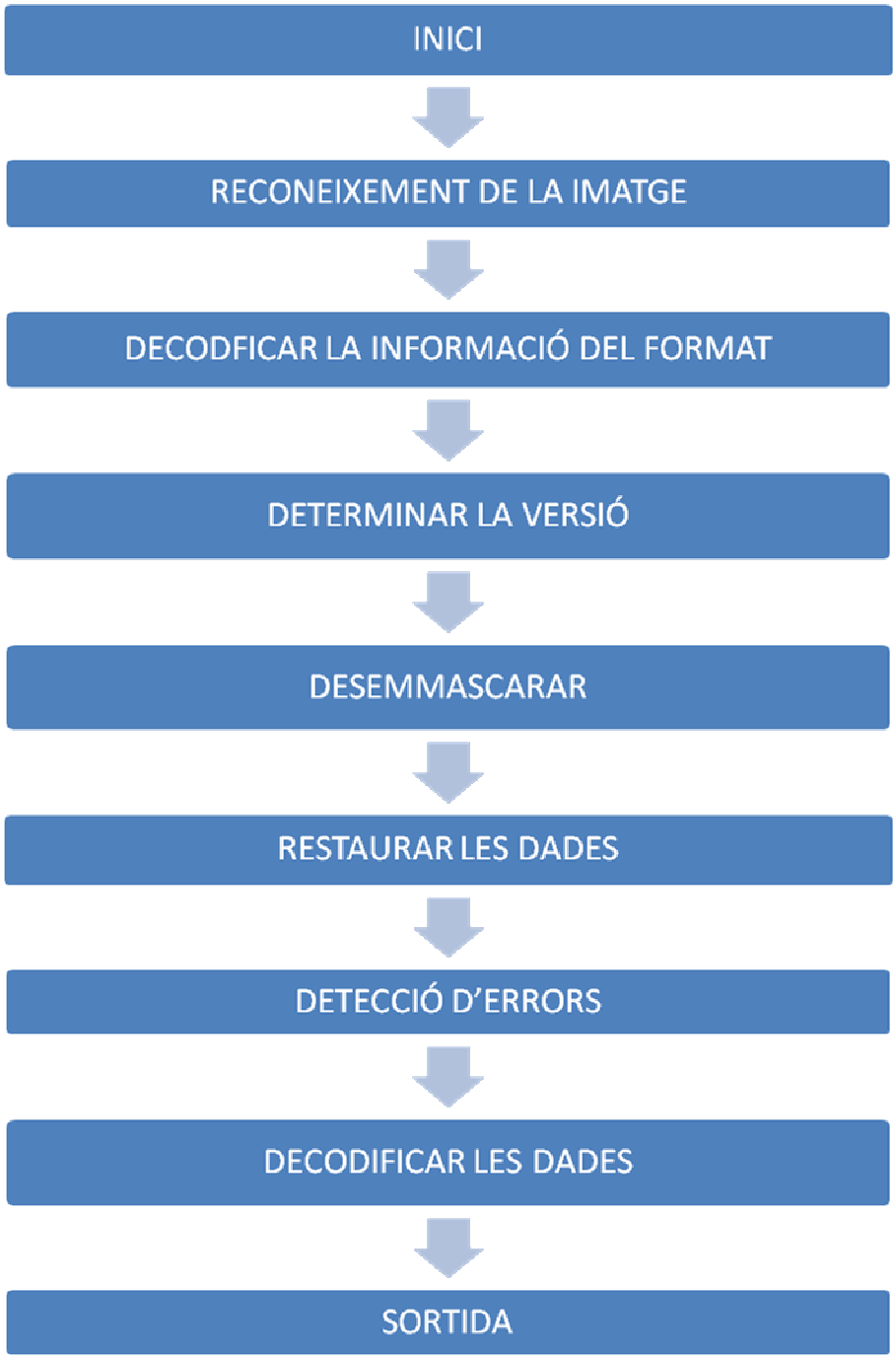
```
public void actionPerformed(ActionEvent e) {
    if(e.getActionCommand()==boto.getActionCommand()){
        String informacio=text.getText();
        codificador(informacio,ecl(),mida(),mida());
    }
    if(e.getActionCommand()==boto2.getActionCommand()){
        int seleccio=seleccionador.showSaveDialog(this);
        if(seleccio==seleccionador.APPROVE_OPTION){
        try {
            ImageIO.write(imatgeCodificada, "png", seleccionador.getSelectedFile());
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        }
    }

}
```

Ja tenim el nostre codificador acabat.

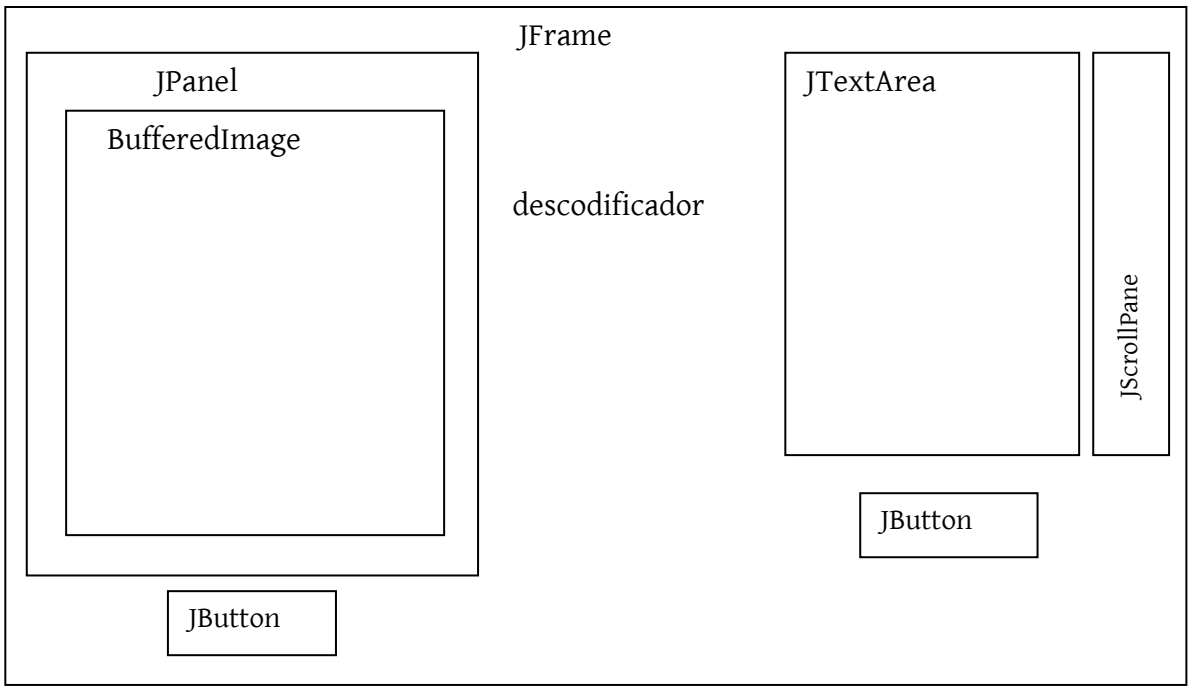
6. COM ES PROGRAMA UN PROGRAMA DECODIFICADOR DE CODIS QR

L'objectiu d'aquest programa es senzill. Elegir una imatge que contingui un codi qr i processar-ho per obtenir la informació codificada . La informació resultant s'ha de mostrar en la pantalla

6.1 DIAGRAMA DE DESCODIFICACIÓ



6.2 ESQUEMA VISUAL DEL PROGRAMA



En aquest programa no utilitzarem la llibreria ZXING encara que tingui el material necessari per a descodificar un codi QR. Aquesta vegada aprofitarem les classes de la llibreria QRCode de la pàgina http://qrcode.sourceforge.jp/, una pàgina japonesa en la que es pot compartir material informàtic amb altres usuaris. Aquesta llibreria és de codi obert i per tant no hi ha cap restricció que ens prohibeixi utilitzar-la en el nostre descodificador. Com l'anterior programa codificador començarem programant la classe que descodifiqui la imatge. Aquesta classe constarà d'un mètode que cridarà a la seva vegada a una classe de la llibreria que realitzarà la majoria de passos per aconseguir la descodificació, però primer, mirem-nos els passos que segueix per fer-ho.

1. Obtenció del símbol

getQRCodeSymbol()

2. Llegeix la imatge

ImageReader

3. Processa la imatge

processImage()

4. Troba el patró de referència

findFinderPatterm()

5. Fa la alineació

findAlignmentPattern()

6. Amb la alineació anterior troba els 0 i 1

getSamplingGrid(FinderPattern,AlignmentPattern)

7. Amb aquests crea la matriu

getQRCodeMatrix(samplingGrid)

8. Descodifica els símbols i obté les dades d'aquest

getDataByte(qrCodeSymbol)

9. Aplica la correcció d'errors

correctDataBlocks()

10. Desfà la matriu amb trames

getDecodeByteArray

11. Llegeix les trames i obté el missatge

6.3 PROGRAMA DESCODIFICADOR DE CODIS QR

El nostre mètode constarà d'un paràmetre *BufferedImage* que serà la imatge a codificar. Al crear el mètode, dintre, instanciem la classe *QRCodeDecoder* que serà la classe encarregada de descodificar la imatge. Per a què aquesta classe descodifiqui una imatge, la imatge ha de tenir implementada una interfície *QRCodeImage* que s'encarrega d'estudiar la mida de la imatge i els píxels. Per aquesta raó crearem una classe paral·lela que implementarà la interfície *QRCodeImage* la qual tindrà un atribut i un paràmetre *BufferedImage* i que en el seu mètode constructor s'encarregarà d'instanciar aquest objecte.

```
public class AnalitzarImatge implements QRCodeImage{
    private BufferedImage imatge;

    public AnalitzarImatge(BufferedImage imatge){
        this.imatge=imatge;
    }

    public int getHeight() {
        return imatge.getHeight();
    }

    public int getPixel(int x, int y) {
        return imatge.getRGB(x, y);
    }

    public int getWidth() {
        return imatge.getWidth();
    }

}
```

Al crear aquesta classe ja podem fer ús del mètode decode() de la classe QRCodeDecoder que ens demana com a paràmetre un objecte QRCodeImage que instanciarem dintre del parèntesi per estalviar-nos la creació d'un objecte apart. Aquesta classe retorna un vector de bytes que contenen la informació descodificada que posteriorment igualarem a un atribut String.

```
public void descodificar(BufferedImage imatge){
    QRCodeDecoder descodificador=new QRCodeDecoder();
    try{
byte[] byteDescodificat=descodificador.decode(new AnalitzarImatge(imatge));
stringDescodificat=new String(byteDescodificat);
    }catch(DecodingFailedException e){
        e.printStackTrace();
    }
}
```

Com és possible que la descodificació ens doni un error encerclem el procés de descodificació dintre d'un try i catch.

Per a descodificar una imatge, primer, l'usuari l'ha de seleccionar i per aquesta raó és imprescindible l'objecte JFileChooser que s'encarregarà del procés. Després de seleccionar la imatge aquesta la representarem sobre un llenç per a què es vegi abans de ser descodificada i donar a l'usuari la possibilitat d'elegir un altra si li sembla.

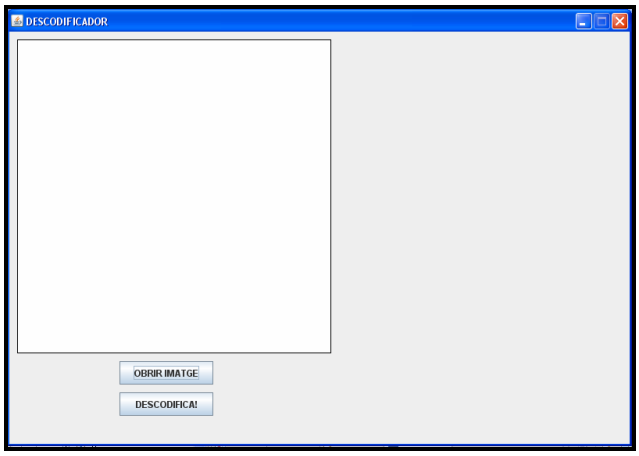
Formem un llenç com el creat en el programa codificador i l'instanciem en la classe descodificadora que implementarà la classe JFrame per a ser una finestra. Per tant en aquesta classe introduïm les comandes necessàries per a què es formi una finestra que contingui un llenç.

```
public class Llenç extends JPanel{
    private Descodificador d;

    public Llenç(Descodificador d){
        this.d=d;
        this.setSize(402,402);
    }

    public void paintComponent(Graphics g){
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, 401, 401);
        g.setColor(Color.BLACK);
        g.drawRect(0,0,401,401);

    }
}
```

Les mides no són iguals a l'hora de pintar els quadrats perquè així el segon no és tapat per cap objecte i pot marcar la vora del llenç en la finestra.

```
public Descodificador(){
    super("DESCODIFICADOR");
    setSize(800,560);
    setLayout(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setResizable(false);
    setVisible(true);
    Llenç llenç=new Llenç(this);
    add(llenç);
    llenç.setBounds(10,10,402,402);
}
```

Creem els objectes JButton com a atributs(els botons per a iniciar la descodificació i per obrir la imatge), els instanciem en la finestra, els adherim i els assignem un lloc i una mida. Es fa el mateix amb l'objecte JFileChooser però aquest ni s'adhereix ni se li assigna un lloc. Implementarem la interfície ActionListener per a utilitzar el seu mètode per obrir el JFileChooser des del botó, per tant necessitarem posar al botó d'obrir en mode de escolta.



Realitzem els mateixos passos que varem realitzar en el programa codificador per a aconseguir la ruta de la imatge a obrir des de l'objecte JFileChooser, però aquesta vegada no l'instanciarem com una finestra per guardar objectes, sinó per a obrir-los.

```
public void actionPerformed(ActionEvent e) {
    if(e.getActionCommand()==obrir.getActionCommand()){
        int seleccio=seleccionador.showOpenDialog(this);
        if(seleccio==seleccionador.APPROVE_OPTION){
            try{
                imatge=ImageIO.read(seleccionador.getSelectedFile());
            }catch(Exception ex){
                ex.printStackTrace();
            }
        }
    }
}
```

Com que ja tenim la ruta de la imatge, introduïm en el mètode *paintComponent()* del llenç la comanda que la representi si l'atribut imatge de la classe finestra en té un. És a dir, quan l'atribut imatge de la classe finestra contingui una imatge, aquesta es representi en el llenç. Per poder representar-la en el mig del llenç, sigui de la mida que sigui, s'han d' introduir una sèrie de càlculs. Aquests càlculs es basen en indicar a l'ordinador que el punt origen de la imatge és la resta de la mida horitzontal de la meitat del llenç menys la mida horitzontal de la meitat de la imatge pel punt x i la meitat de la mida vertical del llenç menys la meitat de la mida vertical de la imatge pel punt y.

```
g.drawImage(d.getImatge(),200-(d.getImatge().getWidth()/2),
        200-(d.getImatge().getHeight()/2),null );
```

Però poden haver casos en què l'usuari vulgui descodificar imatges que superin la mida del llenç per aquests casos, redimensionarem les imatges abans de representar-les, introduint dos paràmetres apart dels utilitzats per indicar la posició, que seran paràmetres que indiquin la mida.

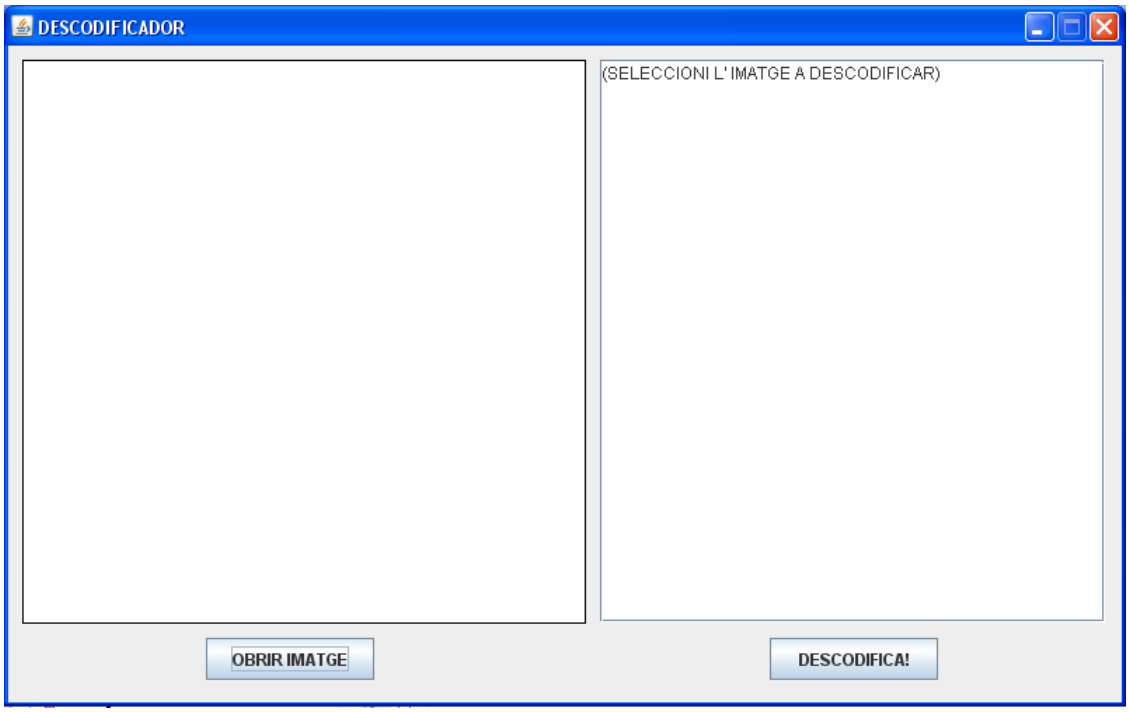
```
if(d.getImatge()!=null){
     if(d.getImatge().getWidth()<400&&d.getImatge().getHeight()<400){
     g.drawImage(d.getImatge(),200-(d.getImatge().getWidth()/2),
              200-(d.getImatge().getHeight()/2),null );
     }
     else{
          g.drawImage(d.getImatge(),1,  1,400,400,null);
     }
}
```

Com que ja tenim programat la part d'obtenció d' una imatge, podem comprovar que el mètode *descodificar(BufferedImage)* funciona. Per a fer-ho, instanciem el mètode en els de la interfície, i indiquem que quan es premi el botó de la finestra el programa iniciï la descodificació i el resultat es mostri en la pantalla de la consola. El resultat es l'espera't.

Només falta que el resultat del mètode descodificador es mostri en un objecte *JTextArea*. Per tant crearem un atribut en la finestra amb aquest objecte i l'instanciarem en el mètode constructor. Per a què l'usuari a l'obrir el programa sàpiga que és el que ha de fer, introduirem un text amb instruccions utilitzant el mètode *setText(String),* a més indicarem que estableixi un salt de línea automàtic quan el text descodificat sigui més llarg que una línea . Si el codi a descodificar és massa llarg necessitarem un objecte JScrollPane per desplaçar-nos pel text, per tant també tindrem que instanciar-lo, tenint cura d'introduir l'objecte JTextArea com a paràmetre. Bloquegem l'objecte JScrollPane i li donem permís per desplaçar-se només verticalment. Després l'adherim i li assignem una mida i una posició.

```
text=new JTextArea();
text.setLineWrap(true);
text.setText("(SELECCIONI L' IMATGE A DESCODIFICAR)");
scroll=new JScrollPane(text);
scroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
add(scroll);
scroll.setBounds(422,10,360,401);
```

Per últim indiquem després de posar en marxa el mètode descodificador que es mostri el resultat en l'objecte JTextArea . Finalment el programa tindrà un aspecte semblant a aquest:



7. HIPOTESIS D'AUGMENT DE LA CAPACITAT D'EMMAGATZEMATGE D'UN CODI QR

Com hem vist en apartats anteriors, la capacitat d'emmagatzematge d'un codi QR depèn de la quantitat de codewords que en te. Per modificar aquesta capacitat haurem de trobar la manera d'aprofitar millor la disposició d'aquest codewords en l'espai i és per aquesta raó que sorgeixen les següents hipòtesis.

1. Reduint la mida del codi obtindrem major espai per poder representar més codewords.

2. Canviant la forma en què estan representats els llindars blancs i negres utilitzant signes (+, | ,- ,) que bàsicament el que farien seria superposar dos codis (un amb llindars: -, i l'altre amb llindars |,) crearien un codi en el qual podríem distingir cadascun dels codis i després codificar-ho.

3. Utilitzant matrius policromàtiques podríem representar més d'un codi en la mateixa imatge. És a dir assignar colors diferents que representin un codi QR diferent.

7.1 DISCUTIM LES HIPÒTESIS:

Si duguéssim a terme la 1a hipòtesi ens adonaríem que al reduir la mida d'un codi QR no s'augmenta la capacitat d'emmagatzematge, tan sols reduïm l'espai per a poder representar codis de versions de símbol majors.

La 2a hipòtesi augmentaria la capacitat d' emmagatzematge però per fer-ho hauríem de canviar tots els mètodes per detectar el codi QR en qüestió ja que ara els patrons de detecció serien diferents. A més a major versió del símbol més costaria captar una imatge del codi enfocant cadascun dels signes ja que al ser primets es desenfocarien de seguida.

La 3a hipòtesi tindria èxit si el programa filtrés correctament tots i cadascun dels colors a l'hora de la descodificació de forma que en la imatge només apareguin els que s'utilitzaren en la codificació.

8. PROGRAMA CREADOR DE CODIS QR AMB CAPACITAT D'EMMAGTZEMATGE AUGMENTADA

A partir de la 3a hipòtesis crearem un programa capaç de codificar 2 codis QR en un. Per a fer-ho podrem utilitzar totes les classes del programa codificador de codis QR que anteriorment s'ha programat, diferint en el mètode codificador que ara tindrà un seguit de canvis que permetran la policromatització de la matriu. El mètode codificador seguirà tenint els mateixos paràmetres (String, ErrorCorrectionLevel, int, int) però els atributs de la classe contindran una matriu de bits extra.

```
private BitMatrix bm1, bm2 ;
```

Dintre del mètode codificador instanciarem dos objectes del tipus QRCode. Seguidament indicarem al programa que divideixi la informació en 2 amb la comanda substring prèviament havent-nos assegurat que la informació a codificar és múltiple de 2, si no és així el mètode afegirà un espai al final de la informació.

```
if((informacio.length()%2)==1){
    informacio=(informacio+" ");
}
```

Ara indicarem que es creïn els respectius codis QR per a cada meitat d'informació i posteriorment que aquests codisQR omplin una matriu de bits amb la mida que especifiquem en els paràmetres. En aquest pas tornem a utilitzar el mètode per ampliar la mida que en el seu moment varem programar.

```
Encoder.encode(informacio1,ecl, qr1);
Encoder.encode(informacio2, ecl, qr2);
bm1=ampliarMida(qr1,amplada,alçada);
bm2=ampliarMida(qr2,amplada,alçada);
```

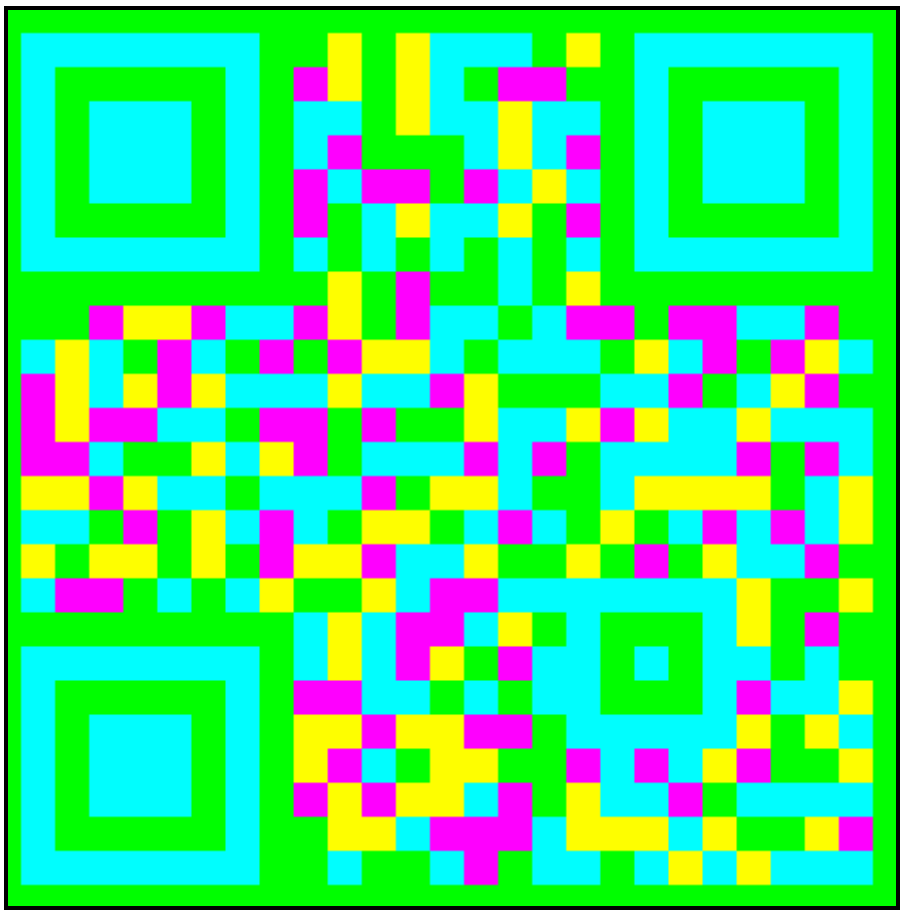
La part més important del programa és el següent procediment. En aquest haurem d'assignar un color diferent per a cada llindar de la matriu resultant. És a dir, si en la imatge final un llindar estar ocupat pel mòdul negre d'una matriu i el mòdul blanc d'un altre assigni un tipus de color, si està ocupat per mòduls negres assigni un color. Per tant hem d'utilitzar un color per a cada possibilitat i en total hi ha 2^2

possibilitats diferents. Aquest procediment es el següent en llenguatge de programació.

```
if((bm1.width==bm2.width)&&(bm1.height==bm2.height)){
for(int y=0;y<bm1.getHeight();y++){
    for(int x=0;x<bm1.getWidth();x++){
        if(bm1.get(x, y)==true&& bm2.get(x, y)==true)
            imatgeCodificada.setRGB(x, y, -16711681);
        else if (bm1.get(x, y)==true&& bm2.get(x, y)==false)
            imatgeCodificada.setRGB(x, y,-256 );
        else if (bm1.get(x, y)==false&& bm2.get(x, y)==true)
            imatgeCodificada.setRGB(x, y, -65281);
        else if (bm1.get(x, y)==false&& bm2.get(x, y)==false)
            imatgeCodificada.setRGB(x, y, -16711936);
    }
}
}
```

*-16711681 =Cyan, -256=Groc , -65281= Magenta , -16711936= Verd

Comproven com codifica el nou codi QR introduint informació a l'atzar i la imatge resultant és la següent.



9.PROGRAMA DESCODIFICADOR DE CODIS QR AMB CAPACITAT D'EMMAGATZEMATGE AUGMENTADA

Realitzar un programa codificador sense un altre anàleg de descodificació no té sentit. Per a descodificar els codis creats en l'apartat anterior utilitzarem el descodificador que varem programar per a codis QR normals. Del programa només realitzarem canvis en el mètode descodificador(BufferedImage). En aquest mètode instanciarem la classe QRCodeDecoder 2 vegades i després 2 imatges. Després seguint el patró utilitzat per omplir els nous codis QR realitzarem el procés invers per transformar-lo en els 2 codis inicials. Per tant les 2 imatges contindran llindars blancs o negres segons el color de la imatge a descodificar.

```
for (int y=0;y<imatge.getHeight();y++){
    for(int x=0;x<imatge.getWidth();x++){
        if(imatge.getRGB(x, y)==-16711681 ||imatge.getRGB(x, y)==-256)
            imatge1.setRGB(x, y, -16777216);
        if(imatge.getRGB(x, y)==-65281||imatge.getRGB(x, y)==-16711936)
            imatge1.setRGB(x, y, -1);
        if(imatge.getRGB(x, y)==-16711681||imatge.getRGB(x, y)==-65281)
            imatge2.setRGB(x,y,-16777216);
        if(imatge.getRGB(x, y)==-16711936|imatge.getRGB(x, y)==-256)
            imatge2.setRGB(x, y, -1);
    }
}
```

Una vegada realitzades les dues imatges les descodifiquem cadascuna per separat com ho fèiem en el descodificador anterior.

```
try{
byte[] byteDescodificat1=descodificador.decode(new AnalitzarImatge(imatge1));
stringDescodificat1=new String(byteDescodificat1);

byte[] byteDescodificat2=descodificador2.decode(new AnalitzarImatge(imatge2));
stringDescodificat2=new String(byteDescodificat2);

}catch(DecodingFailedException e2){
    e2.printStackTrace();
}
```

Una vegada obtingut el string descodificat l'exportem al objecte JTextArea concatenant els dos Strings.

```
if(e.getActionCommand()==traduir.getActionCommand()){
    descodificar(imatge);
    text.setText(stringDescodificat1+stringDescodificat2);
}
```

10. CONCLUSIONS

En l'elaboració d'aquest treball hem assolit els objectius principals que varem proposar. Hem aconseguit realitzar 1 programa per a la codificació d'informació en un codi QR implementant la llibreria ZXING i que es comunica visualment amb l'usuari sense la necessitat d'alterar el codi per a modificar la mida i el nivell de correcció d'aquests. Paral·lelament s'ha realitzat un programa que desxifra la informació d' un codi QR qualsevol i mostra el resultat en pantalla.

En diferents parts del treball s'han realitzat tractaments d'imatge que han estat de gran utilitat per comprovar si es podia dur a terme la 3a hipòtesi. La utilització de matrius policromàtiques aconsegueix augmentar la capacitat d'emmagatzematge del codi. D'aquesta manera podem aprofitar millor l'espai designat per a cadascun dels 2 codis sense renunciar a l'estructura interna que caracteritza un codi QR. A més aquest augment no es limita a duplicar l' informació sinó que podríem augmentar-ne encara més la capacitat d'emmagatzematge si utilitzéssim més colors.

El programa que codifica codis QR modificats duplica la informació emmagatzemada en un codi QR convencional. És a dir, on abans cabia un codi QR ara en caben 2. Aquesta nova característica implica major temps emprat en la codificació ja que ara s'han de codificar dos codis. Aquesta variació és inapreciable perquè l'escala del temps segueix sent de milisegons. Encara que s'hagi augmentat la capacitat d'emmagatzematge s'ha reduït considerablement la capacitat de detecció per part del programa anàleg de descodifcació. Els colors de la imatge han de coincidir exactament amb els indicats en el programa de codificació, i per aquesta raó un codi d'aquest tipus fotografiat es més difícil de descodificar perquè la llum i diversos agents externs impedeixen la netedat de la imatge. El programa descodificador té èxit si la imatge està completament pura i sense soroll extern.

Per aquesta raó proposo possibles treballs de recerca futurs sobre filtrats d'imatge i sobre una propietat molt interessant que assoleixen aquest nous codis QR. Si no els utilitzéssim per augmentar la informació podríem augmentar la seva seguretat fins el punt en que més del 50% de codi és podria reconstruir a partir d'un codi trencat .

11. ANNEX

Hi ha un CD adjunt que conté tots els programes creats fins ara. Per a obrir-los es imprescindible tenir instal·lat JAVA(SE) TM PLATFORM BINARY que es pot instal·lar des de la web oficial de JAVA.

12. BIBLIOGRAFIA

LLIBRES

FERNÁNDEZ ORDÓÑEZ, Ginés Miguel. *Generador e intérprete QR code*. Escola tècnica superior d'enginyeria informàtica de Sevilla.

CRUZ CAMPRUBÍ, Carles. *Aplicació per gestionar codis de barres bidimensionals del tipus QR Code utilitzan un telèfon mòbil* .Universitat Politècnica de Barcelona.

NIEMEYER, Patrick. Curso de Java/Patrick Niemeyer y Jonathan Knudsen. Madrid: Anaya Multimedia, cop.2000 .

VÍDEOS AUTODIDÀCTICS.

http://www.youtube.com/user/niktutos