LAB4: DOCUMENTATION

Maria Beili Mena Dorado (u199138)
Mireia Pou Oliveras (u198721)

1. Description of the problem

Aquesta quarta pràctica consisteix a implementar un disseny basat, principalment, en l'herència i la reutilització de codi. Hem hagut d'implementar diverses classes que des de fora sembla que no tinguin massa relació, però que anirem reutilitzant per implementar-ne d'altres.

Un cop la pràctica ha estat acabada, s'hauria de visualitzar, a la terminal, un seguit de vectors i matrius que comproven que el codi funcioni. →

```
[1.0, 2.0, 3.0]
[0.0, 0.0, 0.0]

Matrix:
[1.0, 0.0]
[0.0, 1.0]

Matrix:
[0.0, 0.0]
[0.0, 0.0]

[1.0, 0.0, 0.0]
[6.123233995736766E-17, 1.0, 0.0]
ERROR!: incompatible dimensions
```

Per poder fer la tasca hem hagut d'implementar 7 classes amb els seus diferents mètodes:

1. Vector
2. Matrix: que està relacionada amb Vector per la relació de composició, ja que una matriu té vers. Per tant, la classe Matrix tindrà un array de Vectors.
3. AudioBuffer: hereta de Vector i representa una seqüència de valors de la d'oscil·lació de l'ona (negatius o positius).
4. Frame (classe abstracta): matriu amb nombres positius, per això hereta de Matrix i poden ser blanc/negre o de color.
5. BWFrame: hereta de Frame.
6. ColorFrame: també hereta de Frame i té declarat els components de cada color que són: Red, Green i Blue.
7. Main: és on cridem a les funcions de cada classe per saber si les hem executat correctament.

Per cada classe hem hagut de definir mètodes:

<u>Sets/Gets</u>

Hem creat els mètodes get() i set() corresponents per a afegir un valor o retornar-lo, sigui a un atribut o un array, a totes les classes exceptuant la d'AudioBuffer i Frame (aquesta última és una classe abstracta i els gets/sets es defineixen a les classes filles).

```
public void set(int i, int j, double val){
    values[i].setVec(j,val);
}

public double get(int i, int j){
    return values[i].getVec(j);
}
```

A les classes que hereden d'altres classes, per a fer els sets()/gets() cridem als mètodes del pare. Posem un exemple de BWFrame:

```java
public void setBW(int i, int j, double val){
    super.set(i, j, val);
}

public double getBW(int i, int j){
    return super.get(i, j);
}
```

Vector

Hem implementat els seus atributs (es troben al diagrama) i els següents mètodes (juntament amb els gets()/sets()):

- Constructor on li passem el paràmetre de dimensió de Vector, inicialitzem l'array de valors i cridem a la funció d'inicialitzar-los a 0.
- Set_zeros() → posa tots els components d'un vector a 0. Aquesta funció podria estar definida com a privada, però nosaltres l'hem posat pública, ja que sinó no es pot veure des del main.
- multiplyVec() → multiplica el vector per l'escalar passat per paràmetre. Per a fer-ho utilitzem un "for".
- printVec() → imprimieix un vector. Tots els print que hi ha és perquè agafi la forma que volem.
- set3D() → assigna els valors passats per paràmetre a les posicions corresponents del vector. És com 3 set() de l'apartat anterior fets a la vegada.
- multiplyMat() → mutiplica un vector per una matriu. Primer, hem comprovat que la longitud del vector sigui igual al nombre de files de la matriu, ja que sinó no es pot fer el càlcul. Un cop sabem que les dimensions són compatibles, creem un vector que agafa els valors del vector que volem multiplicar per la matriu. Això ho fem perquè el vector de veritat va canviant els valors a mesura que fem les operacions. Finalment, fem un "for" per recórrer les files i columnes de la matriu i anem fent les operacions necessàries.

```java
public void multiplyMat(Matrix m){
    if(values.length != m.getrows()){
        System.out.println("ERROR!: incompatible dimensions");
    }else{
        Vector multiply_vector = new Vector(d: 3);
        multiply_vector.set3DVec(values[0], values[1], values[2]);
        for(int i=0; i<m.getrows(); i++){
            double value = 0;
            for (int j=0; j<m.getcols(); j++){
                value += multiply_vector.getVec(j)*m.get(i,j);
            }
            values[i] = value;
        }
    }
}
```

Matrix

La matriu consisteix en un array de vectors. Té dos atributs per definir les seves dimensions: les files i les columnes. Per a treballar amb ella, hem definit els següents mètodes (i els get / set del primer apartat):

- Constructor: Li passem dos paràmetres, n i m, que s'atribueixen a les files i les columnes. A més, dintre el constructor inicialitzem l'array de vectors i, per a cada vector, inicialitzem el vector cridant el seu constructor. En aquest cas, no cridem a set_zeros_matrix() perquè el constructor de cada vector ja els inicialitza a 0.
- MultiplyMat(): Funció que multiplica la matriu amb un escalar passat per referència. Per a fer-ho, utilitzem un bucle "for" que agafa cada vector de l'array. Per a cada vector cridem a la funció de multiplicar el vector amb l'escalar.
- PrintMatrix(): Imprimeix la matriu. Igual que abans, amb un bucle "for" imprimim cada vector cridant la funció de printVec().
- Set_zeros_matrix(): Posa 0 a cada valor de la matriu. De la mateixa manera que als altres mètodes, creem un bucle que itera sobre la llista de vectors i, per a cada un, crida a la funció set_zeros() de vector.
- create3DRotationMat(): En aquesta funció, on se li passa un paràmetre alpha, girarem la matriu alpha graus. Per a fer-ho seguim el dibuix que hi ha a l'explicació de la pràctica. Com alpha ha d'estar en radiants, primer fem una espècie de comprovació on mirem si alpha és 90 i, si ho és, el passem a radiants. Després calculem el cosAlpha i el sinAlpha i afegim els valors corresponents a cada espai de la matriu amb la funció set3DVec.

```
public void create3DRotationMat(double alpha){
    if (alpha == 90){
        alpha = 90*180/Math.PI;
    }

    double cosAlpha = Math.cos(alpha);
    double sinAlpha = Math.sin(alpha);

    values[0].set3DVec(cosAlpha, -sinAlpha, k: 0);
    values[1].set3DVec(sinAlpha, cosAlpha, k: 0);
    values[2].set3DVec(i: 0, j: 0, k: 1);
}
```

Frames

Classe abstracta que hereta de Matrix i que, per tant, no necessita atributs. Pel seu constructor utilitzem super() per passar els paràmetres, ja que els atributs estan definits a la classe pare. No necessitem definir el mètode changeBrightness() perquè és abstracte. Ho farem a les seves classes filles.

1. BWFrame (hereta de Frame)

No té cap més atributs que els de la classe pare (que realment ara són els de Matriu). Per a treballar amb la classe hem definit els següents mètodes:

- Constructor: Igual que el constructor de Frame. Simplement, li passem els paràmetres i cridem al constructor del pare.
- Set/get explicat al primer apartat.

```
//Constructor
public BWFrame(int n, int m){
    super(n,m);
}
```

- Mètode changeBrightness: Com que BWFrame hereta de Frame i aquesta última és una classe abstracta, això implica que ha d'implementar tots els seus mètodes abstractes. Per a fer-ho s'ha d'escriure @override a sobre el mètode. Dintre seu simplement cridem al mètode de multiplicar la matriu pel paràmetre delta (que canvia la brillantor) amb la paraula super.

```
@Override
public void changeBrightness(double delta){
    super.multiplyMat(delta);
}
```

2. Color Frame

Igual que BW Frame, tampoc té cap atribut més que els de la classe pare (Matriu). Per a treballar amb la classe hem definit els següents mètodes:

- Constructor: Igual que el constructor de BW Frame.
- Set/Get: en aquest cas el set/get és una mica més complicat, ja que treballa amb 3 paràmetres en comptes de un. En el cas de set, li passem els paràmetres r,g,b i utilitzem la funció RGBToVal per a passar-ho a valor. En el cas de get és al contrari, obtenim un valor i utilitzem la funció valToRGB per a passar-ho als paràmetres.

```
public void setColor(int i, int j, int r, int g, int b){
    double ret = RGBToVal(r, g, b);
    super.set(i,j,ret);
}

public int[] getColor(int i, int j){
    double rgb = super.get(i, j);
    int[] color = valToRGB(rgb);
    return color;
}
```

- ChangeBrightness(): Igual que a BW Frame, hem d'implementar els mètodes abstractes de Frame fent un @override. El mètode, per això, és una mica diferent. Fa servir dos "for" per a iterar sobre la matriu i agafa els valors r,g,b de cada posició amb la funció getColor(). Nosaltres utilitzem la funció RGBToVal() per a passar-lo a un valor i poder-lo multiplicar per delta. Finalment el tornem a guardar.

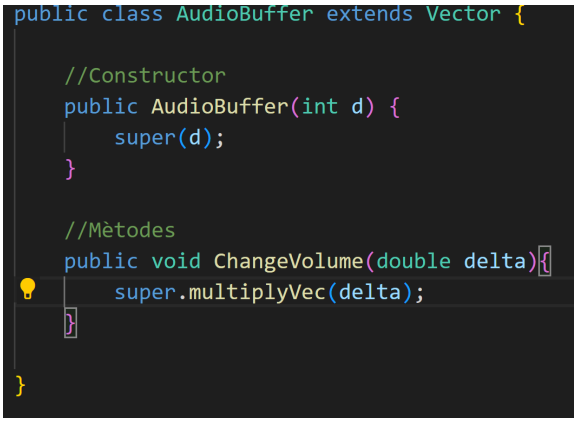
```
@Override
public void changeBrightness(double delta) {
    for (int i=0; i<super.getrows(); i++){
        for (int j =0; j<super.getcols(); j++){
            int[] rgb = getColor(i,j);
            double val = RGBToVal(rgb[0], rgb[1], rgb[2]);
            val = val*delta;
            super.set(i,j,val);
        }
    }
}
```

- ChangeRGB(): Probablement la funció més complexa de la classe. Comença igual que changebrigthness(): itera sobre cada posició de la matriu i agafa els valors r,g,b. Un cop els tenim, sumem a cada un d'ells el respectiu paràmetre passat a la funció (canvia el color). Després fem servir un altre "for" per iterar sobre r,g,b i comprovar que estiguin entre 0 i 255. Finalment, tornem a posar els valors a la funció.

```
public void changeRGB(int dR, int dG, int dB){
    for (int i=0; i<super.getrows(); i++){
        for (int j =0; j<super.getcols(); j++){
            int[] rgb = getColor(i,j);
            rgb[0] = rgb[0]+dR;
            rgb[1] = rgb[1]+dG;
            rgb[2] = rgb[2]+dB;
            for (int z=0; z<rgb.length;z++){
                if (rgb[z] > 255){
                    rgb[z] = 255;
                }
                if (rgb[z] < 0){
                    rgb[z] = 0;
                }
            }
            setColor(i, j, rgb[0], rgb[1], rgb[2]);
        }
    }
}
```

AudioBuffer

La classe AudioBuffer hereta de Vector i no necessita cap més atributs que els seus. Al constructor fem servir super() per passar els paràmetres (perquè els atributs estan guardats a la classe pare). El seu mètode ChangeVolume() simplement multiplica els valors del vector per un nombre real positiu més gran o petit d'1 per incrementar o disminuir. Per fer això hem usat la funció multiplyVec().



2. Alternative solution discussed

En aquesta pràctica hem tingut tres dubtes principals.

1. **Vector:** No sabíem exactament com declarar l'array de valors de vector. Vam pensar a fer-ho amb LinkedList o ArrayList, però la cosa es complicava molt. Al principi, declarant-ho com un double[] no ens funcionava, però vam trobar el problema i vam procedir a fer-ho d'aquesta manera.
2. **Vector - multiplyMat():** Després de provar la funció el primer cop no funcionava. Vam estar molta estona mirant amb el "Debug" que era el que la feia fallar fins que ens vam adonar que necessitàvem un vector auxiliar (nosaltres l'hem anomenat multiply_vector) perquè el vector anava canviant els seus valors i això feia que només el primer funcionés.
3. **Color Frame - ChangeBrightness():** Al principi havíem pensat d'implementar aquesta funció igual que a BW Frame; és a dir, cridant a la funció multiplyMat(delta) i ja està. Després ens vam adonar que estàvem treballant amb 3 paràmetres (red,green,blue) en comptes d'un. Així doncs, ho vam fer d'una altra forma.

3. Conclusion

En resum i mirant-la globalment, creiem que la pràctica ha estat ben implementada. Els mètodes i els atributs han estat fets tal com consta a l'enunciat i el test ha tingut resultats positius.

És cert i en som conscients, però, que no hem tingut en compte cap mena d'error a l'hora de l'input. És a dir, si una funció necessita un "int" i li passem una "Vector" la pràctica donarà error i ja està, no hi haurà cap opció per canviar-ho ni al codi s'ha implementat res per evitar-ho.

Tots els problemes que hem tingut estan explicats al punt 2.