Marina Castellano
Mar Moreno

INFORME LAB 4: IMPLEMENTING DIFFERENT CLASSES OF A DESIGN THAT USES INHERITANCE

En aquest laboratori hem implementat les classes Vector, Matrix, AudioBuffer, Frame, BWFrame i ColorFrame, que vam dissenyar durant el seminari 4 i dintre de cada una hem implementat els atributs i mètodes necessaris. L'objectiu és crear les classes que formen part del projecte i les relacions que hi han entre elles, i després imprimir els resultats en un nova classe, classe que hem anomenat TestMatrix.

CLASSES

- **Vector:** Els atributs d'aquesta classe son: values, que es un double[] i dimension que és un int.
  Els mètodes son: Vector, al qual li passem la dim (int). Dintre de Vector definim dim i values, com a un double de dimension.

```
public Vector(int dim){
    this.dimension = dim;
    values = new Double[dimension];

}
```

  El segon mètode es set, al qual li passem i, i val, assignant així el valor a la posició que tenim com a paràmetres.

```
public void set(int i, double val) {
    values[i] = val;

}
```

  A la funció 3D vector, li passem i,j,k. Fem un if comparant la longitud de values a 3, si es compleix assignem a l'array la posició 0,1,2 a i,j,k.

```java
public void set3D(double i, double j, double k){
    if(values.length == 3){
        values[0] = i;
        values[1] = j;
        values[2] = k;
    }

}
```

Al mètode multiplyVectValue li passem double s. Dintre de la funció fem un for que va iterant, si el valor de i és més petit que la longitud de values, retorna el valor de values[i] * s.

```java
public void multiplyVectValue(double s){
    for(int i= 0; i<values.length; i++){
        values[i] *= s;
    }

}
```

A la funció getValor li passem int i, i retorna els values d'aquesta de la posició i.

```java
public Double getValor(int i){
    return values[i];
}
```

```java
ublic void matrixMultiply(Matrix matrix){

  if(values.length != matrix.getRows()){
      System.out.print(s: "Esta operación no es possible");
  }else{
      Vector vector = new Vector(matrix.getRows());
      vector.zero();

      for(int j = 0; j < matrix.getCols(); j++){
          double suma= 0;
              for(int k= 0; k < matrix.getRows(); k++){
                  double matval = matrix.getMatrix(j, k);
                  double vectval = values[k];
                  suma += matval*vectval;
              }
              vector.set(j, suma);
          }
          this.setValues(vector.getValues());
      }

  }
```

MatrixMultiply: Li passem Matrix matrix.

Dintre de MatrixMultiply creem un nou vector, al qual li passem matrix.getRows(). GetRows() es una funció que hem creat en la classe Matrix. Fem un for que itera mentres j sigui més petit que les columnes de matrix. Inicialitzem suma i després tornem a fer un for que itera

si k és més petita que les files de matrix. Dintre d'aquest for definim matval, que es la matriu j (columnes), k(files), que hem creat; i també definim vectval, que els valors de les files, és a dir el vector. Fem la suma de la multiplicació de matval i vectval. Fora d'aquest segon for fem un vector.set(j, suma), per a guardar en el vector j (files) i suma, que es la multiplicació del vector per la matriu. I fora dels dos fors, fem un setValues, per guardar els valors del vector.

SetValues: Li passem Double[] values. Dintre de setValues difinim values.

```
public void setValues(Double[] values) {
    this.values = values;
}
```

GetValues, que retorna els values.

```
public Double[] getValues() {
    return values;
}
```

Fem un PrintVector, per a després cridar a aquesta funció des de TestMatrix,i que imprimeix els vectors que hem creat.

```
public void PrintVector(){
    System.out.print(s: "[");
    for(int i= 0; i<dimension; i++){
        System.out.print(values[i]);
        if(i<dimension-1){
            System.out.print(s: ",");
        }
    }
    System.out.print(s: "]");
    System.out.print(s: "\n");
}
```

Hem creat una funció que posi tots els valor del vector a zero, per a utilitzar en altres funcions.

```
public void zero() {
    //possa a zero tots els valors del vector
    for(int i = 0; i < dimension; i++){
        values[i] = (double) 0;
    }
}
```

- **Matrix:** Els atributs de matrix son: valores, rows i cols.Dintre del constructor Matrix, al qual li hem passat int n i int m, hem definit les files i les columnes igualant-les a las variables adequades. Després hem creat valores, que es un vector de columnes i hem fet un for que itera mentres i < longitud de valores, on igualem la variables valores de la iteració corresponen a un nou vector que es de files.

```
public Matrix(int n, int m){
    this.rows = n;
    this.cols = m;
    valores = new Vector[m];
    for(int i=0;i<valores.length;i++){
        valores[i] = new Vector(n);
    }
}
```

Per implementar el setMatrix hem utilitzat el mètode set de la classe Vector agafant el vector en la posició i li passem j i valor.

```
public void setMatrix(int i, int j, double valor){
    valores[i].set(j, valor);
}
```

En el getMatrix, hem utilitzat també el getValor de vector, i li hem passat j.

```
public Double getMatrix(int i, int j){
    return valores[i].getValor(j);
}
```

Aquest mètode retorna valores, aquesta funció l'utilitzem en la classe Colorframe, en la funció changeRGB.

```
public Vector[] getValores() {
    return this.valores;
}
```

Al mètode multiplyVectValue li passem double s. Dintre de la funció fem un for que va iterant, si el valor de i és més petit que les files, i a dintre d'aquest un for un altre for que va iternant si j és més petit que les columnes. Si es compleix, retorna un double de v, que agafa per cada final el valor de la columna. I fa un set per guardar el valor de la columna i la multiplicació de la matriu.

```
public void MultiplyMatValue(double s){

    for(int i = 0; i < rows; i++){
        for(int j= 0; j < cols; j++ ){
            double v = valores[i].getValor(j);
            valores[i].set(j, v*s);
        }
    }
}
```

A continuació hem implementat el mètode Create3DRotationMat el qual crea una matriu 3x3 que fa que quan la multipliques per un vector columna de dimensió 3 roti per l'angle amb el qual li passes com a paràmetre del mètode. Primer hem comprovat que la longitud dels vectors siguin de 3 perqué coincideixin les dimensions. Després per cada fila hem anat assignant els valors corresponents.

```
public void Create3DRotationMat(double alpha){
    if(valores.length == 3){
        valores[0].set(i: 0, Math.cos(alpha));
        valores[0].set(i: 1, -Math.sin(alpha));
        valores[0].set(i: 2,val: 0);
        valores[1].set(i: 0,Math.sin(alpha));
        valores[1].set(i: 1, Math.cos(alpha));
        valores[1].set(i: 2,val: 0);
        valores[2].set(i: 0,val: 0);
        valores[2].set(i: 1,val: 0);
        valores[2].set(i: 2,val: 1);

    }
}
```

Els mètodes getRows i getCols els hem creat per a que retornin les columnes i files, aquestes funcions les hem utilitzat en MatrixMultiply de la classe vector.

```java
public int getRows() {
    return rows;
}

public int getCols() {
    return cols;
}
```

El mètode PrintMatrix les creat, per a que imprimeixi les matrius que hem creat en la classe testMatrix.

```java
public void PrintMatrix(){

    for(int k=0; k < valores.length; k++){
        valores[k].PrintVector();
    }
    System.out.print(s: "\n");
}
```

Hem creat una funció que posi tots els valor de la matriu a zero, per a utilitzar en altres funcions.

```java
public void zeroM(){
    for(int l = 0; l < valores.length; l++){
        valores[l].zero();
    }
}
```

- **AudioBuffer:** La classe AudioBuffer no té atributs perquè és hereditària de la classe de Vector.

  Els mètodes d'aquesta classe son el constructor i changeVolum.

  Al constructor AudioBuffer li hem passat d, que es la dimensió, i hem fet un super(d).

  En la funció changeVolum, al qual li hem passat delta (double). Dintre de la funció hem fet un if, que si delta == 0, significa que no s'ha modificar el volumen, pero que si delta és diferent de 0 (else), multipliqui delta amb la funció que hem creat en Vector, multiplyVectValue.

```java
public void changeVolume(double delta){
    if (delta == 0){
        System.out.print(s: "no se ha incrementado el volumen");
    } else{
        this. multiplyVectValue(delta);
    }
}
```

- **Frame:** Aquesta classe igual que l'anterior no té atributs ja que és hereditària de la classe Matrix. En el constructor li passem els atributs de la classe de la que hereda. Com veiem apart de posar extends Matrix hem de posar abstract, ja que aquesta classe és una abstracta que té un mètode abstracte changeBrightness, el qual no hem implementat ja que sera implementada per les classes que heredin d'aquesta, que son: BWFrame i ColorFrame.

```java
public abstract class Frame extends Matrix{


    public Frame(int n, int m){
        super(n, m);

    }

    public abstract void changeBrightness(double delta);


}
```

- **BWFrame:** Com hem mencionat a la classe anterior, aquesta és hereditària d'ella. En el constructor utilitzem la paraula clau super per passar-li els atributs.

```java
public class BWFrame extends Frame{

    public BWFrame(int n, int m){
        super(n, m);

    }
```

Després hem implementat els métodes set i get de la classe. En el set hem utilitzat el mètode de set de la classe Matrix amb els paràmetres que li passa a la funció. Per el get també cridem a la funció get de la classe Matrix i per convertir el valor double que ens retorna la funció, hem hagut de convertir el getMatrix en int ja que és el que havia de retornar aquesta funció.

```
    public void Set(int i, int j, int val){
        this.setMatrix(i, j, val);

    }

    public  int getBW(int i, int j){
        int result = (int)getMatrix(i, j);
        return result;
    }
```

Per últim, hem implementat la funció heredada de frame, changeBrightness. Dintre d'aquesta funció hem fet un procediment molt semblant al de changeVolum, hem multiplicat el valor de delta per una matriu.

```
@Override
public void changeBrightness(double delta) {
    // TODO Auto-generated method stub
    if (delta == 0){
        System.out.print(s: "no se ha incrementado el brillo");
    } else{
        this.MultiplyMatValue(delta);
    }

}
```

- **ColorFrame:** Aquesta igual que l'altre, que ja hem explicat, és una herencia de la classe Frame. El primer que hem fet ha sigut escriure els mètodes privats que ens proporcionava la pràctica, el valToRGB i el RGBToVal.

```
public class ColorFrame extends Frame {

    private int[] valToRGB(double rgb){
        int[] ret = new int[3];
        ret[0] = ((int) rgb >> 16) & 255;
        ret[1] = ((int) rgb >> 8) & 255;
        ret[2] = ((int) rgb) & 255;
        return ret;
    }


    private double RGBToVal(int r, int g, int b){
        double ret = (r << 16) | (g << 8) | b;
        return ret;
    }
```

En el constructor hem utilitzat la paraula clau super per passar els atributs de la superclasse.

```java
public ColorFrame(int i, int m){
    super(i, m);
}
```

A continuació hem implementat els mètodes set i get. Per el setRGB li passem 5 paràmetres de tipus int, la fila (i) i la columna (j) de la que sera la matriu, i els números dels colors vermell, verd i negre, r, g, b respectivament. Creem una variable double que será el número que representin els colors i després creem una matriu cridant el setMatrix passant el paràmetres corresponents sent els valors els números de colors.

En el getRGB li hem passat els paràmetres i i j. Fem un return utilitzant la funció que hem creat anteriorment, valToRGB,

```java
public void setRGB(int i,int j , int r, int g, int b){
    double ret = RGBToVal(r, g, b);
    setMatrix(i, j, ret);

}


public int[] getRGB(int i, int j){
    return this.valToRGB(this.getValores()[i].getValues()[j]);
}
```

L'últim mètode que hem implementat en la classe ColorFrame es changebrightness, el procediment és molt semblant al de changeVolum, multipliquem una matriu pel vector delta.

```java
@Override
public void changeBrightness(double delta){
    // TODO Auto-generated method stub
    if (delta == 0){
        System.out.print(s: "no se ha incrementado el brillo");
    } else{
        this.MultiplyMatValue(delta);
    }
```

- **TestMatrix:** Aquesta és la classe és on s'executa el programa a partir de les funcions implementades en les demés classes. En aquesta classe podrem mostrar

per pantalla la informació que volem comprovar i si el codi que hem implementat funciona correctament.

El primer que comprovem és si la classe vector funciona correctament creant un vector nou, passant-li valors cridant a les funcions corresponents i després mostrant-lo per pantalla. Després hem cridat a la funció que multiplica un vector per un escalar i i ho hem imprès per pantalla. Veiem que el resultat ha sigut l'esperat.

```
Vector v = new Vector(dim: 3);
v.set(i: 0, val: 1);
v.set(i: 1, val: 2);
v.set(i: 2, val: 3);
v.PrintVector();
v.multiplyVectValue(s: 4);
v.PrintVector();
v.zero();
v.PrintVector();
```

```
[1.0,2.0,3.0]
[4.0,8.0,12.0]
[0.0,0.0,0.0]
```

De la mateixa comprovem que la matriu es pugui crear correctament i veiem que si per el resultat obtingut.

```
Matrix m = new Matrix(n: 2,m: 2);
m.setMatrix(i: 0, j: 0, valor: 1);
m.setMatrix(i: 0, j: 1, valor: 0);
m.setMatrix(i: 1, j: 0, valor: 0);
m.setMatrix(i: 1, j: 1, valor: 1);
m.PrintMatrix();
m.MultiplyMatValue(s: 2);
m.zeroM();
m.PrintMatrix();
```

```
[1.0,0.0]
[0.0,1.0]

[2.0,0.0]
[0.0,2.0]

[0.0,0.0]
[0.0,0.0]
```

Per últim comprovem que els mètodes set3D del vector, Create3DRotationMat de la matriu i el matrixMultiply funcionen correctament i també veiem que si.

```
Vector vector = new Vector(dim: 3);
vector.set3D(i: 1, j: 0, k: 0);

Matrix ma = new Matrix(n: 3, m: 3);
ma.Create3DRotationMat(Math.PI/2);
ma.PrintMatrix();

vector.PrintVector();
vector.matrixMultiply(ma);
vector.PrintVector();
```

```
[0.0,-1.0,0.0]
[1.0,0.0,0.0]
[0.0,0.0,1.0]

[1.0,0.0,0.0]
[0.0,1.0,0.0]
```

Ara volem comprovar si el codi implementat en la classe Audiobuffer és el correcte. Per veure si és correcte creem una instància d'Audiobuffer amb un valor aleatòri, creem un vector de 3 dimensions i cridem al mètode changeVolume per a canviar-li el volum. Després fem un print del vector per veure si és correcte el funcionament. Comprovem que funciona correctament ja que ha multiplicat els valors del vector per el valor que volem canviar el volum.

```
//AudioBuffer
AudioBuffer a = new AudioBuffer(d: 3);
a.set3D(i: 2,j: 3,k: 4);
a.changeVolume(delta: 6);
a.PrintVector();
```

CONCLUSIÓ

Una vegada implementat el codi hem observat que el resultat ha estat l'esperat, ja que s'han imprès be les matrius i els resultats. Només hem pogut comprovar el funcionament de les classes vector, matriu i Audiobuffer, ja que no hem fet la part opcional de la pràctica que és on es veu si el funcionament és el correcte.

Ens hem trobat amb alguns problemes durant la pràctica, que hem estat capaç de resoldre. Principalment hem tingut problemes a l'hora de fer el mètode changebrightness en les classes BWFrame i ColorFrame. No sabíem com s'havia d'implementar la funció exactament, ja que no enteníem si havíem de multiplicar el valor delta per una matriu, semblant al changeVolum, o havíem de modificar el valor de delta semblant a com ho hem fet en el changeRGB. Com no teníem el testImage per comprovar-ho, ho hem deixat de la següent forma:

```
    this.MultiplyMatValue(delta);
```

Creiem que aquest laboratori ens ha ajudat molt ha veure com funcionen mètodes que no havíem utilitzat anteriorment.