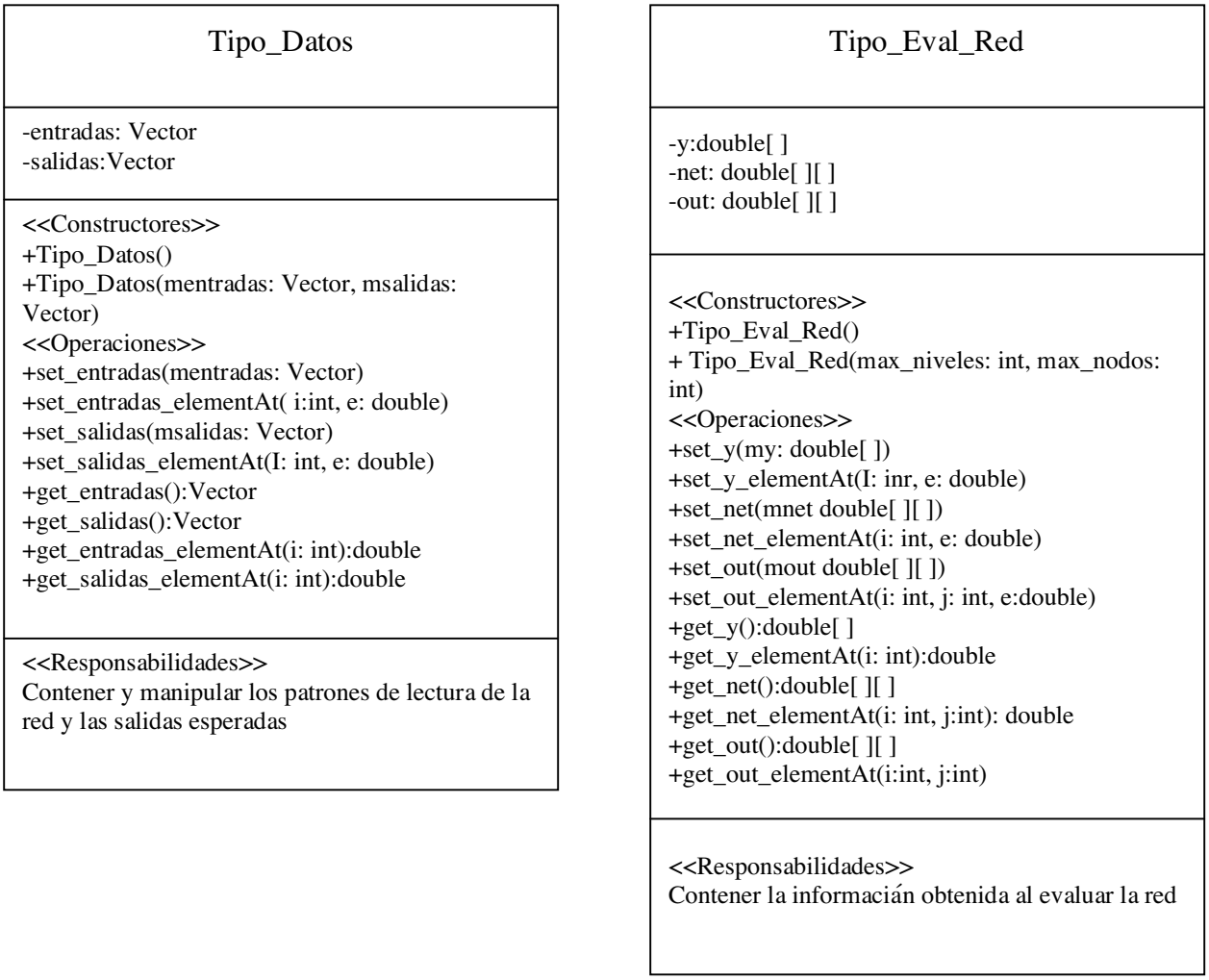
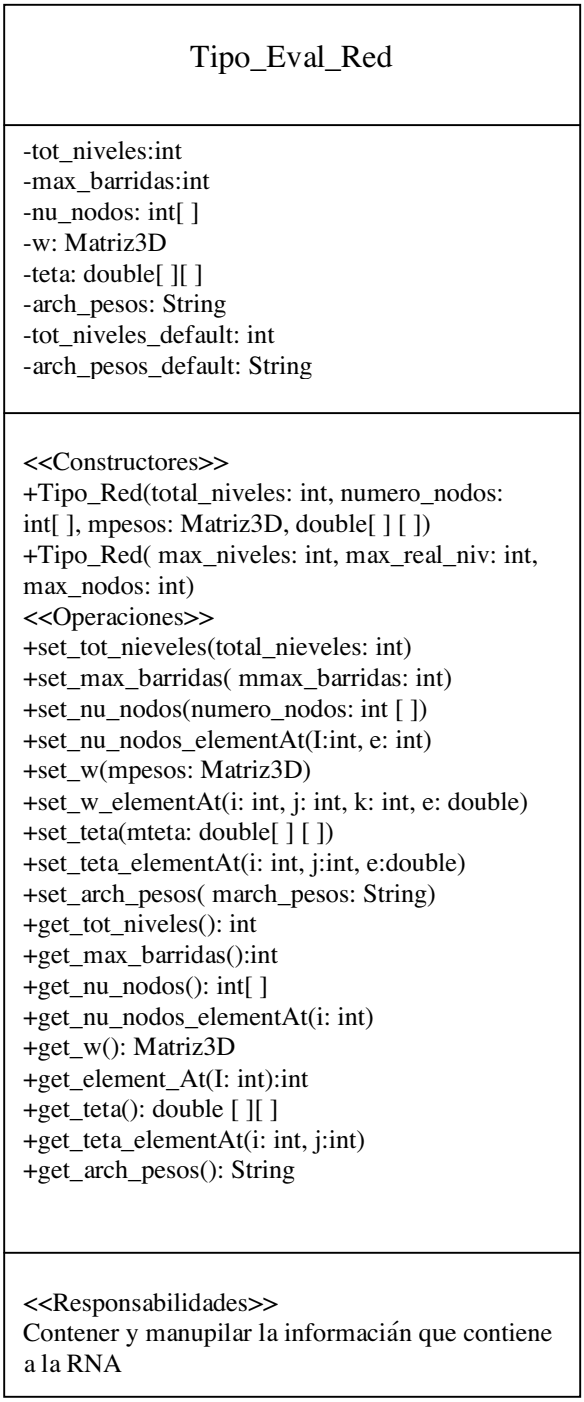
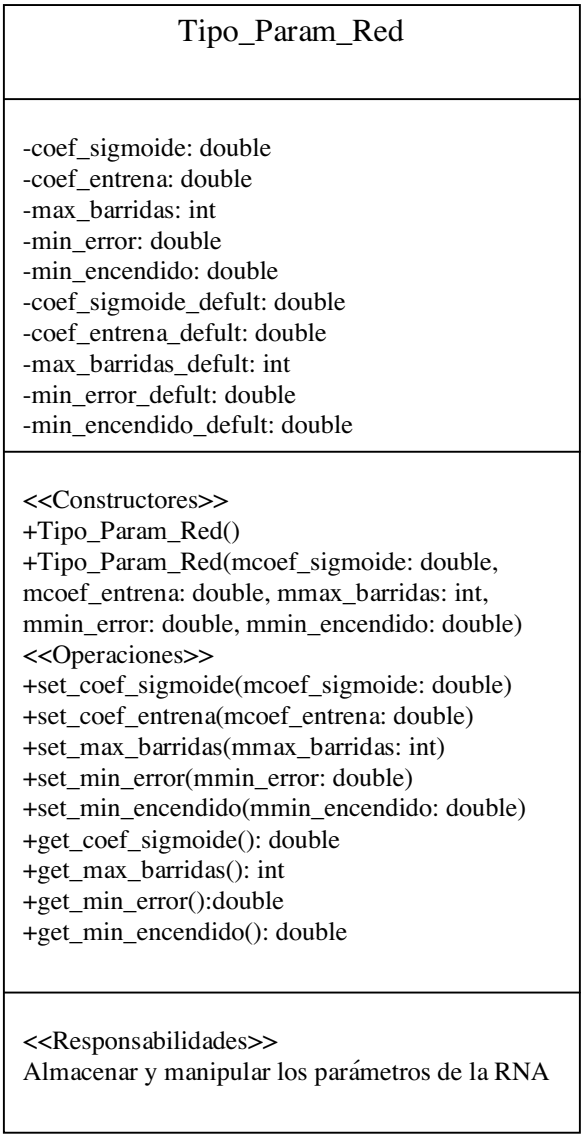
Apéndice B

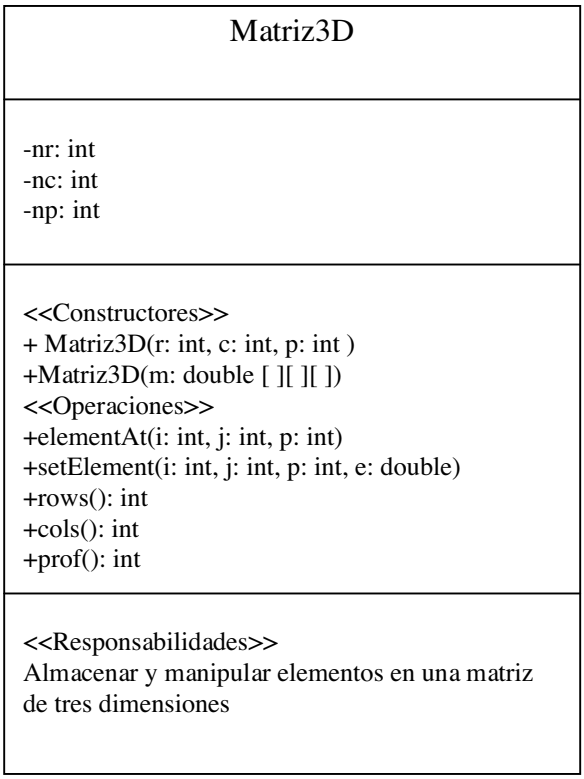
B.1 Diagramas de Clase de la Red Neuronal Artificial

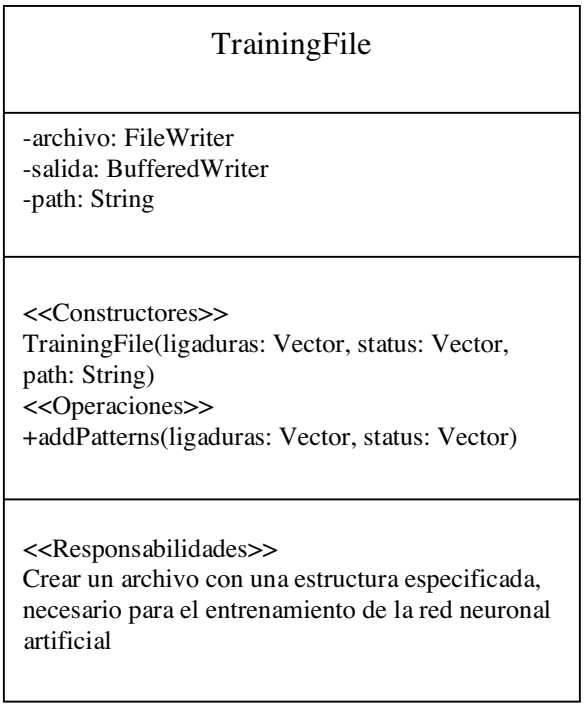
Red
-Max_Real_Niv: int -Max_Nodos: int -Max_Niveles: int
<<Constructor>> +red(r : int, n : int, o : int) <<Operaciones>> +aprende(:Tipo_Param_Red, :Tipo_Red, r:Tipo_Eval_Red, deseo:String, cam_archt :String, cilcos:int, texto: textArea) +clasifica(p: Tipo_Param_Red, r: Tipo_Red) -compara(sal_real: double[][], dato: Tipo_Datos, tot_salidas: int):int -evalua-red(dato: Tipo_Datos, r: Tipo_Red, e: Tipo_Eval_Red, coef_sig:double, wish:String) -gen_arch_entrena() -gen_ran() -graba_pesos(r:Tipo_Red) -lee_patrones(arch_ent: String, tot_entradas:int, tot-salidas:int) -lee_pesos(r: Tipo_Red) +modifica_parametros(p: Tipo_Param_Red) -modifica_pesos(d: Tipo_Datos, e: Tipo_Eval_Red, r: Tipo_Red, coef_ent: double, coef_sig: double) +modifica_red(r: Tipo_Red) -sigmoid(x:double, lambda:double): int -sigmoide_prima(x:double, lambda:double): int -suma_delta(lay:int, nodo:int, r: Tipo_Red, delta: double[][]):int
<<Responsabilidades>> Comportarse como una red neuronal artificial, aprendiendo patrones y clasificándolos posteriormente.


Tipo_Datos
-entradas: Vector
-salidas:Vector
<<Constructores>>
+Tipo_Datos()
+Tipo_Datos(mentradas: Vector, msalidas: Vector)
<<Operaciones>>
+set_entradas(mentradas: Vector)
+set_entradas_elementAt(i:int, e: double)
+set_salidas(msalidas: Vector)
+set_salidas_elementAt(I: int, e: double)
+get_entradas():Vector
+get_salidas():Vector
+get_entradas_elementAt(i: int):double
+get_salidas_elementAt(i: int):double
<<Responsabilidades>>
Contener y manipular los patrones de lectura de la red y las salidas esperadas
Tipo_Eval_Red
-y:double[ ]
-net: double[ ][ ]
-out: double[ ][ ]
<<Constructores>>
+Tipo_Eval_Red()
+ Tipo_Eval_Red(max_niveles: int, max_nodos: int)
<<Operaciones>>
+set_y(my: double[ ])
+set_y_elementAt(I: inr, e: double)
+set_net(mnet double[ ][ ])
+set_net_elementAt(i: int, e: double)
+set_out(mout double[ ][ ])
+set_out_elementAt(i: int, j: int, e:double)
+get_y():double[ ]
+get_y_elementAt(i: int):double
+get_net():double[ ][ ]
+get_net_elementAt(i: int, j:int): double
+get_out():double[ ][ ]
+get_out_elementAt(i:int, j:int)
<<Responsabilidades>>
Contener la informacián obtenida al evaluar la red


Tipo_Eval_Red
-tot_niveles:int
-max_barridas:int
-nu_nodos: int[ ]
-w: Matriz3D
-teta: double[ ][ ]
-arch_pesos: String
-tot_niveles_default: int
-arch_pesos_default: String
<<Constructores>>
+Tipo_Red(total_niveles: int, numero_nodos: int[ ], mpesos: Matriz3D, double[ ] [ ])
+Tipo_Red(max_niveles: int, max_real_niv: int, max_nodos: int)
<<Operaciones>>
+set_tot_nieveles(total_nieveles: int)
+set_max_barridas(mmax_barridas: int)
+set_nu_nodos(numero_nodos: int [ ])
+set_nu_nodos_elementAt(I:int, e: int)
+set_w(mpesos: Matriz3D)
+set_w_elementAt(i: int, j: int, k: int, e: double)
+set_teta(mteta: double[ ] [ ])
+set_teta_elementAt(i: int, j:int, e:double)
+set_arch_pesos(march_pesos: String)
+get_tot_niveles(): int
+get_max_barridas():int
+get_nu_nodos(): int[ ]
+get_nu_nodos_elementAt(i: int)
+get_w(): Matriz3D
+get_element_At(I: int):int
+get_teta(): double [ ][ ]
+get_teta_elementAt(i: int, j:int)
+get_arch_pesos(): String
<<Responsabilidades>>
Contener y manupilar la informacián que contiene a la RNA


Tipo_Param_Red
-coef_sigmoide: double
-coef_entrena: double
-max_barridas: int
-min_error: double
-min_encendido: double
-coef_sigmoide_defult: double
-coef_entrena_defult: double
-max_barridas_defult: int
-min_error_defult: double
-min_encendido_defult: double
<<Constructores>>
+Tipo_Param_Red()
+Tipo_Param_Red(mcoef_sigmoide: double, mcoef_entrena: double, mmax_barridas: int, mmin_error: double, mmin_encendido: double)
<<Operaciones>>
+set_coef_sigmoide(mcoef_sigmoide: double)
+set_coef_entrena(mcoef_entrena: double)
+set_max_barridas(mmax_barridas: int)
+set_min_error(mmin_error: double)
+set_min_encendido(mmin_encendido: double)
+get_coef_sigmoide(): double
+get_max_barridas(): int
+get_min_error():double
+get_min_encendido(): double
<<Responsabilidades>>
Almacenar y manipular los parámetros de la RNA

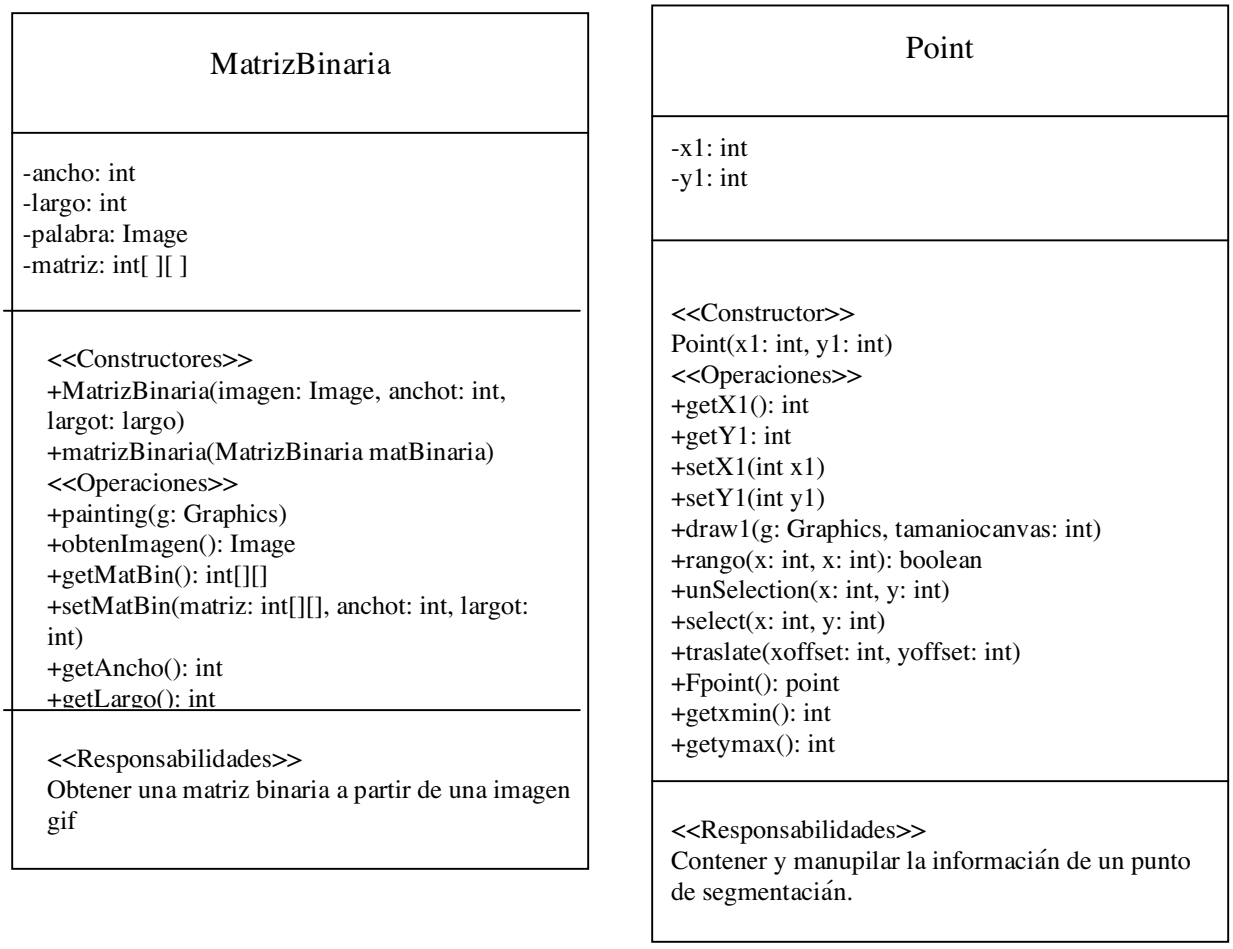

Matriz3D
-nr: int
-nc: int
-np: int
<<Constructores>>
+ Matriz3D(r: int, c: int, p: int)
+Matriz3D(m: double [ ][ ][ ])
<<Operaciones>>
+elementAt(i: int, j: int, p: int)
+setElement(i: int, j: int, p: int, e: double)
+rows(): int
+cols(): int
+prof(): int
<<Responsabilidades>>
Almacenar y manipular elementos en una matriz de tres dimensiones

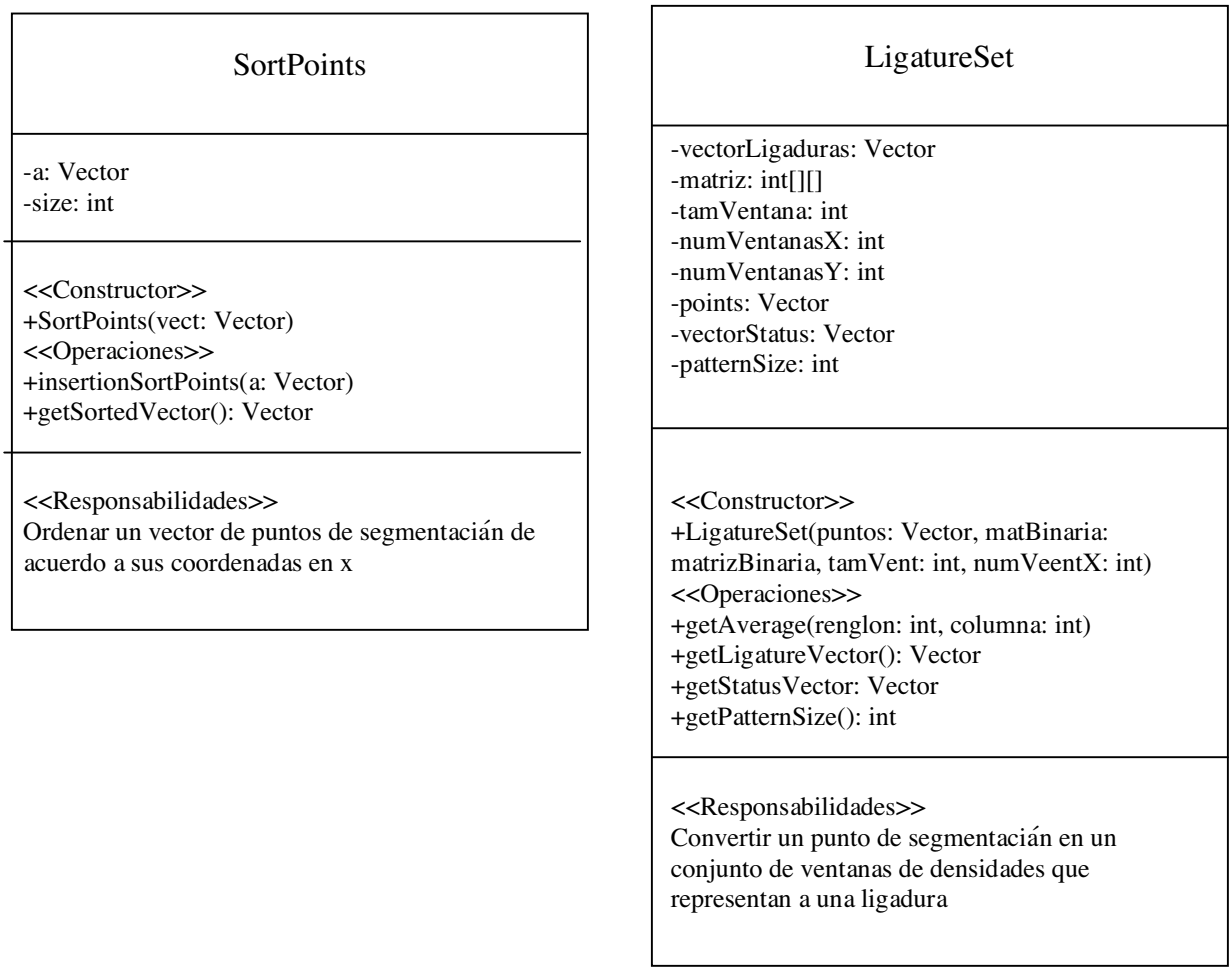

TrainingFile
-archivo: FileWriter
-salida: BufferedWriter
-path: String
<<Constructores>>
TrainingFile(ligaduras: Vector, status: Vector, path: String)
<<Operaciones>>
+addPatterns(ligaduras: Vector, status: Vector)
<<Responsabilidades>>
Crear un archivo con una estructura especificada, necesario para el entrenamiento de la red neuronal artificial

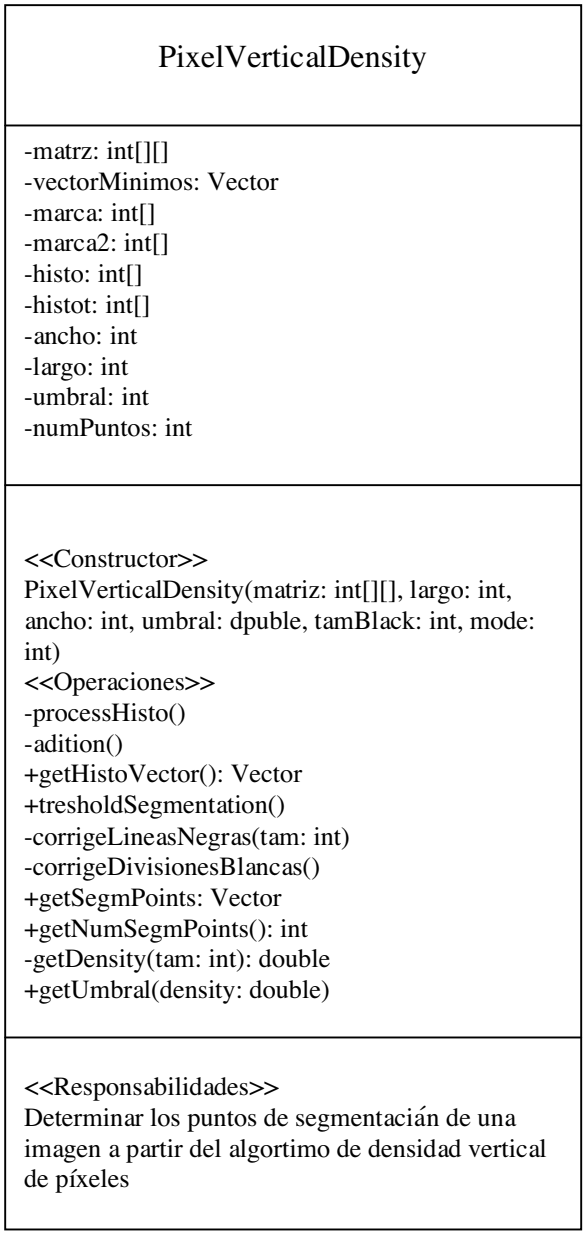
B. 2 Diagramas de Clase de la Herramienta

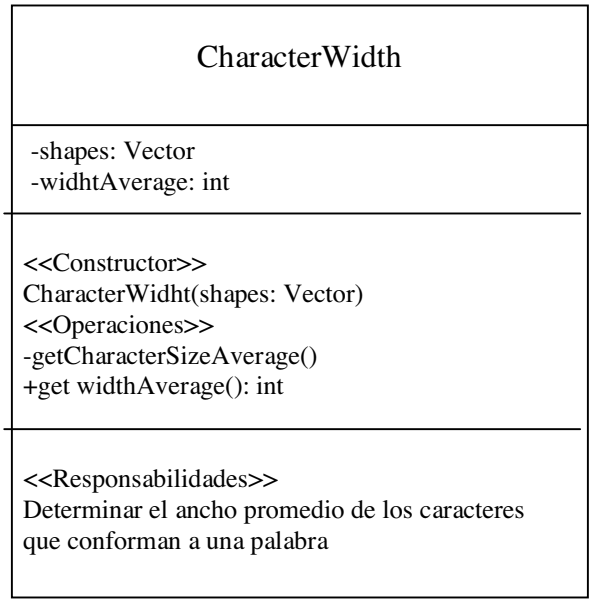
HAWOST
<<Atributos Principales>> -drawing: Drawing -drawing2: Drawing -kit: Toolkit -palabra: Image -red: Red
<<Operaciones principales>> +init() -actionPerformed(event: ActionEvent) -windowClosing(event: WindowEvent) -makeComponents() -setupGUI() -tieEvents() -makeDrawingPanel() -makeOutputPanel() -makeFileDialog() -getPositionScreen(): double[ ] -quit()
<<Responsabilidades>> Proveer y controlar la interfaz gráfica de la aplicacián

Drawing
<<Atributos principales>> -shapes: Vector -selectedShapes: Vector -buffer: Vector -matBin: MatrizBinaria
<<Constructor>> +Drawing(tamx: int, tamy: int) <<Operaciones Principales>> +mouseClicked(event: MouseEvent) +paint(g: Graphics) +setImageData(path: String, kit: Toolkit) +getVerticalMaxMinPixels(matriz: int [][], ancho: int, largo: int) +addPointsToShape(points: Vector) +applyVeticalPixelDensityA() +applyCharacterWidthA() +applyContourA() +copy() +cut() +unselectAll() +reset() +changeSelect()
<<Responsabilidades>> Presentar un area de dibujo en las que se pueda hacer visualizaciánes y ediciones.


MatrizBinaria
-ancho: int
-largo: int
-palabra: Image
-matriz: int[ ][ ]
<<Constructores>>
+MatrizBinaria(imagen: Image, anchot: int, largot: largo)
+matrizBinaria(MatrizBinaria matBinaria)
<<Operaciones>>
+painting(g: Graphics)
+obtenImagen(): Image
+getMatBin(): int[][]
+setMatBin(matriz: int[][], anchot: int, largot: int)
+getAncho(): int
+getLargo(): int
<<Responsabilidades>>
Obtener una matriz binaria a partir de una imagen gif
Point
-x1: int
-y1: int
<<Constructor>>
Point(x1: int, y1: int)
<<Operaciones>>
+getX1(): int
+getY1: int
+setX1(int x1)
+setY1(int y1)
+draw1(g: Graphics, tamaniocanvas: int)
+rango(x: int, x: int): boolean
+unSelection(x: int, y: int)
+select(x: int, y: int)
+traslate(xoffset: int, yoffset: int)
+Fpoint(): point
+getxmin(): int
+getymax(): int
<<Responsabilidades>>
Contener y manupilar la informacián de un punto de segmentacián.


SortPoints
-a: Vector
-size: int
<<Constructor>>
+SortPoints(vect: Vector)
<<Operaciones>>
+insertionSortPoints(a: Vector)
+getSortedVector(): Vector
<<Responsabilidades>>
Ordenar un vector de puntos de segmentacián de acuerdo a sus coordenadas en x
LigatureSet
-vectorLigaduras: Vector
-matriz: int[][]
-tamVentana: int
-numVentanasX: int
-numVentanasY: int
-points: Vector
-vectorStatus: Vector
-patternSize: int
<<Constructor>>
+LigatureSet(puntos: Vector, matBinaria: matrizBinaria, tamVent: int, numVeentX: int)
<<Operaciones>>
+getAverage(renglon: int, columna: int)
+getLigatureVector(): Vector
+getStatusVector: Vector
+getPatternSize(): int
<<Responsabilidades>>
Convertir un punto de segmentacián en un conjunto de ventanas de densidades que representan a una ligadura


PixelVerticalDensity
-matrz: int[][]
-vectorMinimos: Vector
-marca: int[]
-marca2: int[]
-histo: int[]
-histot: int[]
-ancho: int
-largo: int
-umbral: int
-numPuntos: int
<<Constructor>>
PixelVerticalDensity(matriz: int[][], largo: int, ancho: int, umbral: dpuble, tamBlack: int, mode: int)
<<Operaciones>>
-processHisto()
-adition()
+getHistoVector(): Vector
+tresholdSegmentation()
-corrigeLineasNegras(tam: int)
-corrigeDivisionesBlancas()
+getSegmPoints: Vector
+getNumSegmPoints(): int
-getDensity(tam: int): double
+getUmbral(density: double)
<<Responsabilidades>>
Determinar los puntos de segmentacián de una imagen a partir del algortimo de densidad vertical de píxeles


CharacterWidth
-shapes: Vector
-widhtAverage: int
<<Constructor>>
CharacterWidht(shapes: Vector)
<<Operaciones>>
-getCharacterSizeAverage()
+get widthAverage(): int
<<Responsabilidades>>
Determinar el ancho promedio de los caracteres que conforman a una palabra