Universidad de Santiago de chile.
Facultad de ingeniería.
Departamento de ingeniería informática.

Informe de laboratorio 3:

Simulación de una plataforma estilo "Photoshop" en lenguaje JAVA.

Nombre: Javiera Varela.
RUT: 19.957.716-6
Profesor: Roberto Gonzales.
Asignatura: Paradigmas de la programación.

Contenido

1. Introducción. 3
1.1 Objetivos 3
2. Descripción del problema. 3
3. Descripción del Paradigma. 3
4. Análisis del problema 4
5. Diseño de la solución. 4
5.1. Operación para la creación de una imagen. 5
5.2. Operaciones sobre el reconocimiento de imagen. 6
5.3. Operaciones sobre la imagen 6
5.4. Operaciones que no se llevaron a cabo dentro de la imagen. 8
5.5. Diagramas de análisis 9
5.6. Diagrama de diseño. 10
6. Aspectos de la implementación. 11
6.1. Estructura del proyecto: 11
6.2. Compilador o interprete usado: 11
7. Instrucciones de uso. 11
8. Resultados. 12
9. Conclusiones. 13

1. Introducción.

Este informe corresponde al laboratorio número 3 del curso paradigma de la programación. En este se abordará el paradigma orientado a objetos, el cual se basa en el concepto de clases y objetos. Este tipo de programación destaca en emplear una estructurar un software en piezas simples y reutilizables para crear instancias individuales de objetos.

Como herramienta para poder llegar a la solución del problema, se utilizará el lenguaje de programación java a través del IDE Intellij. Algo destacable dentro de este lenguaje es que puede utilizarse como la especificación de un problema, en vez de especificar pasos para llegar a la solución de un problema.

Dentro de este informe se describirá el problema a moldear, las características de este y sus limitaciones, se abordará una mayor descripción del paradigma a trabajar, se analizará el problema planteando sus requerimientos específicos que se deben cubrir, se explicara el diseño de la solución y la lógica detrás de los algoritmos creados para su realización, las instrucciones de uso, los resultados y conclusiones.

1.1 Objetivos

Como objetivos de este proyecto, se busca aprender sobre el paradigma y programación orientada a objetos para futuros usos en diferentes proyectos, además de aprender el uso de herramientas como JAVA y los instrumentos que otorga este lenguaje para la correcta realización del simulador.

2. Descripción del problema.

El problema consta en desarrollar una simulación a una aplicación de edición de imágenes del tipo GIMP o Adobe Photoshop. Se busca la manipulación y edición de imágenes a través de un enfoque del paradigma orientado a objetos.

Este proyecto se debe concentrar en la utilización de imágenes del tipo RGB o RGB-D, Bit y Hexadecimal las cuales constan de ser una imagen con información en el espacio de colores y un espacio tridimensional.

3. Descripción del Paradigma.

El paradigma orientado a objetos se basa en abstracciones de la realidad, los cuales pueden contener entidades que contienen atributos y métodos con el propósito de intercambiar información. El modelamiento de las entidades nos permie interactuar directamente desde el código, existiendo técnicas de herencia, polimorfismo, acoplamiento y encapsulamiento.

El paradigma orientado a objetos tiene algunos elementos importantes en la construcción de un código:

a. Clases: Una clase corresponde a una implementación de un TDA.
b. Objetos: Son representaciones activas o instancias de una clase.
c. Atributos: Tipo de datos que componen a una clase.
d. Métodos: Expresan comportamientos que puede realizar un objeto sobre si mismos u otros objetos.

e. Constructor: No es un elemento por si mismo dentro del código, pues corresponde a un tipo especial de método, que permite indicar los valores iniciales de los atributos. También reserva memoria necesaria para albergar todos los daos del objeto.

4. Análisis del problema.

Como sé indicaba con anterioridad, el laboratorio busca crear una aplicación similar a un editor de imágenes, en el cual se pueda trabajar con este de una manera más simplificada. Para ello debemos enfocarnos en el constructor principal a trabajar, la cual consta de ser la imagen.

Antes de poder aplicar los elementos necesarios para trabajar en la aplicación, debemos definir una estructura base para poder implementar cada uno de los requerimientos básicos del programa. Para ello ser consiente de que es una imagen y en que consiste esta.

La definición técnica de una imagen digital se define como una representación de dos dimensiones basada en una matriz numérica frecuentemente binaria. Teniendo en cuenta esto, la aplicación a crear busca añadir unos requerimientos básicos y específicos que se deben cubrir y deben trabajar con la imagen. Estos constan de ser:

- Recortar una imagen.
- Invertir una imagen horizontalmente.
- Invertir una imagen verticalmente.
- Comprimir una imagen en base a la eliminación de mayor frecuencia.
- Convertir a hexadecimal.
- Visualizar la imagen.
- Rotar la imagen 90° hacia la derecha.
- Histograma.
- Descomprimir una imagen en base a restitución del color con mayor frecuencia.
- Cambiar un píxel dentro de la imagen.
- Invertir el color Bit.
- Invertir el color RGB.
- Transformar una imagen a String.
- Separar una imagen en base a sus capas de profundidad.

Además, se busca añadir una interfaz interactiva que proporcione mayor comodidad al usuario para poder ingresar los datos necesarios dentro de la aplicación.

5. Diseño de la solución.

El proyecto de la simulación de un editor de imágenes en JAVA, al estar basado principalmente en la programación orientada objetos, debemos moldear las clases con las cuales se trabajará.

1) Imagen: Representación de una imagen la plataforma. Para poder crear una imagen se debe tener los datos de anchura, altura y los pixeles.

2) Pixeles: Representación abstracta de los pixeles miembros de la imagen. Para poder crear un pixel se necesita de la coordenada x, coordenada y, profundidad y el tipo de pixel a trabajar
3) Pixelbit: Representación extendida de los pixeles. Contiene un atributo del tipo bit el cual solo acepta enteros del tipo 0 o 1. Dentro de su representación hereda los elementos x, y, además de profundidad.
4) PixelRGB: Representación extendida de los pixeles. Contiene 3 atributos correspondiente a un valor RGB, es decir, tanto R, G y B solo aceptan enteros que estén en el rango de 0 <= RGB <= 255. Hereda los elementos de la representación pixeles.
5) PixelHexadecimal: Representación extendida de los pixeles. Contiene un atributo del tipo String con un largo de 6 caracteres. Hereda los elementos de la representación pixeles.

5.1. Operación para la creación de una imagen.

Para poder implementar cada una de las operaciones, se debe crear el constructor imagen. Este constructor debe tener en consideración 3 datos principales que son otorgados por el usuario:

- Anchura. -> Entero.
- Altura. -> Entero.
- Pixeles. -> Lista.

Esto lograra que cada imagen tenga una estructura de la siguiente forma:

- Imagen = [Anchura, Altura, Pixeles].

Debemos tener en consideración que cada píxel debe tener una estructura propia, pues una imagen puede contener dentro de ella pixeles del tipo binarios, Hexadecimales o RGB. Para ello debemos ser conscientes de que cada píxel cuenta con una clase propia, por lo tanto, se debe implementar constructor que soporte cada una de sus condiciones.

Hay variables que se repiten dentro de cada estructura de un píxel, pues estas son las correspondientes a su ubicación dentro de un plano “(x,y)” y la profundidad correspondiente de esta. Por lo tanto, se debe utilizar una clase pixel el cual debe contener:

- X. -> Entero. Corresponde a la posición X dentro de un plano cartesiano.
- Y. -> Entero. Corresponde a la posición Y dentro de un plano cartesiano.
- Depth. -> Entero. Corresponde a la profundidad de cada pixel dentro de la imagen.

Al encontrar las variables que hacen únicos a cada pixel y lo diferencian según su composición, debemos ser conscientes de que cada uno tendrá su propia clase. Para ello, un pixel del tipo pixbit debe almacenar dentro de sus valores una variable entera que almacene valores del 0 y 1, un pixel del tipo pixhex debe almacenar una estructura del tipo string ya que debe almacenar valores hexadecimales y finalmente un pixel del tipo pixrgb, el cual debe almacenar 3 valores correspondientes al rojo (R), verde (G) y azul (B), los cuales serán un entero que alcance los valores entre 0 y 255.

Como estos elementos, además de sus particularidades, son parte de una estructura más grande, deben heredar los atributos de la clase pixel, quedando de la siguiente forma:

- Pixbit = [X, Y, Bit, Depth]

- PixRGB = [X, Y, Reed, Green, Blue, Depth]
- PixHex = [X, Y, Hexadecimal, Depth]

5.2. Operaciones sobre el reconocimiento de imagen.

Para ciertas acciones y métodos dentro del programa, se debe tener en consideración que tipo de imagen es la que se desea trabajar, pues algunos métodos no permiten imágenes del tipo PixBit, PixRGB o PixHex. Para ello se tiene lo siguiente:

1. **IsBitmap:**
   - Método que recibe como dominio una imagen y revisa las variables dentro del constructor. Da un booleano "True" si la estructura está compuesta por Pixbits.
2. **IsRGBmap:**
   - Método que recibe como dominio una imagen y revisa las variables dentro del constructor. Da un booleano "True" si la estructura está compuesta por PixRGBs.
3. **IsHexmap:**
   - Método que recibe como dominio una imagen y revisa las variables dentro del constructor. Da un booleano "True" si la estructura está compuesta por PixHexs.

Existen otras operaciones sobre el reconocimiento de imagen, que se ven ligados principalmente una vez aplicado un método. Estos casos son representados por:

4. **IsCompress:**
   - Método que recibe como dominio una imagen, revisa si todos los pixeles son iguales. Da un booleano "True" si la estructura está comprimida.

Existen otras operaciones sobre el reconocimiento de imagen, que se ven ligados principalmente una vez aplicado un método. Estos casos son representados por:

5.3. Operaciones sobre la imagen

Estas operaciones pueden ser realizadas por parte del usuario para realizar modificaciones a la estructura imagen:

1. **flipH:**
   - Permite al usuario dar vuelta la imagen de manera horizontal. La lógica tras el diseño de esta solución se basa en la posición de las coordenadas dentro de cada piel. Para ello se sigue la siguiente formula matemática:

     (Anchura – 1) – X = Nueva coordenada X.

     Una vez obtenida la nueva coordenada, esta debe ser remplazada dentro del píxel. Cabe destacar que esta solución es valida para cualquier tipo de imagen.
2. **FlipV:**
   - Sigue la misma lógica que el método flipH, pero en vez de trabajar con el eje X de cada píxel, se utiliza el eje Y. Para ello sigue la siguiente formula:

     (Altura – 1) – Y = Nueva coordenada Y.

3. **Crop:**
   - Método que permite cortar la imagen. Para la utilización de este método, el usuario debe ingresar las coordenadas (x1, y1) y (x2, y2) las cuales generan un rango, donde se conservan los pixeles donde sus coordenadas encajen dentro de estos límites.
4. **imageRGBtoHex:**
   - Método que permite transformar una imagen que contiene PixRGB a una imagen de pixeles PixHex. La lógica detrás de esta solución se basa en la comprensión de cada formato.
     Un PixRGB consta de 3 variables [Red, Green, Blue], los cuales a su vez tienen un valor dentro de las unidades decimales. Por su parte, un PixHex consta de una variable del tipo String que almacena un valor en variables hexadecimales.
     A través de una función nativa de JAVA, cada variable RGB se transforma a un valor Hexadecimal y se guarda dentro de un string. Luego devuelve una lista con nuevo pixeleshex.
5. **imageToHistogram:**
   - Retorna cantas veces se repite un color dentro de una imagen, independiente del tipo de píxel que ese compuesta la imagen.
     Cuando se trata de un pixel bit, solo retorna frecuencias del valor 1 o 0.
     Cuando se trata de un pixel RGB, devuelve cuantos pixeles se repiten con los mismos colores y la frecuencia individual de cada uno.
     En un píxel hexadecimal, devuelve cuantas veces se repite un string.
6. **Rotate90:**
   - Método que permite girar la imagen 90° hacia la derecha. La lógica detrás de este método corresponde en crear nuevas contraseñas que estén presentes dentro del plano de la imagen, es decir, no deben ser negativas.
     Para esto se utiliza la siguiente estructura:

     $$(Y, (Altura - 1) - X)$$

     De esta forma en cada pixel se cambia sus valores (X, Y) y remplazándolo por la estructura antes presentada.
7. **Compress:**
   - Método que permite comprimir una imagen. Su planteamiento se basa en solo almacenar los pixeles con el color que menos se repite dentro de la imagen.
8. **changePixel:**
   - Método que permite cambiar un pixel dentro de una imagen. Para el optimo funcionamiento se necesita establecer las bases necesarias para su uso. Se tomo la decisión de que el método debe verificar el tipo de imagen y solo aceptar pixeles de remplazo de este tipo.
9. **InvertedColorBit:**
   - Método que permite obtener el color simétricamente opuesto en cada canal dentro de un píxel.
     Los colores simétricamente opuestos de cada pixbit se ven representado de la siguiente manera:

     0 -> 1
     1 -> 0

     Por lo tanto, cada bit dentro de la imagen debe ser remplazado por su opuesto.
10. **InvertedColorRGB:**
    - Este método permite obtener el color simétricamente opuesto en cada canal dentro de un píxel.
      Un ejemplo para el color simétricamente opuesto:

0 -> 255
1 -> 254
2 -> 253

Una solución matemática para este caso es tomar el valor máximo correspondiente a 255 y restarle el valor contenido en las variables R, G y B.

pixrgb(1, 1, 190 (R), 0 (G), 234 (B), 255, PD)
255 - 190 = 65.
255 - 0 = 255.
255 - 255 = 0.

De esta forma se planea obtener el valor opuesto del RGB.

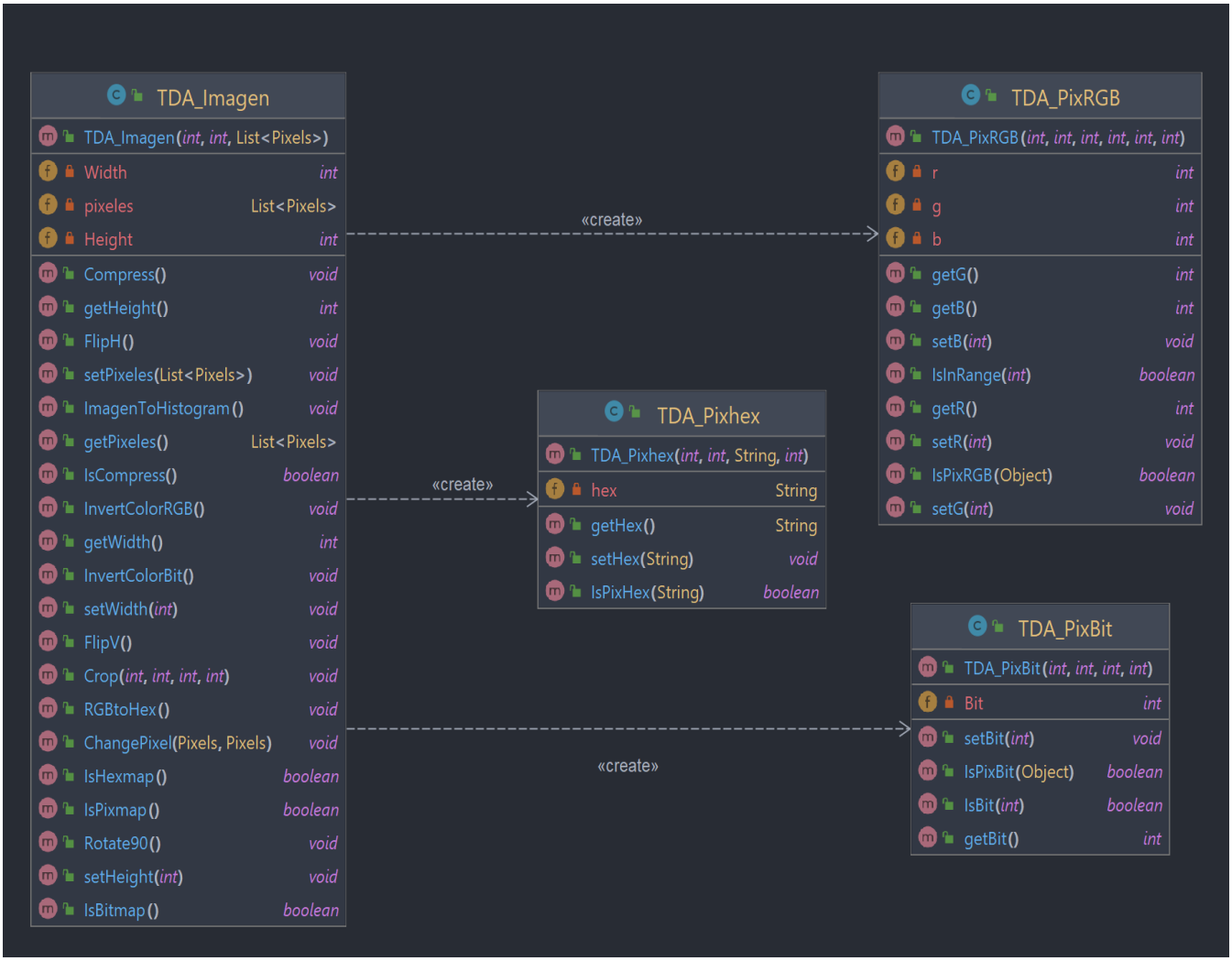
5.4. Operaciones que no se llevaron a cabo dentro de la imagen.

Existen operaciones que no se llevaron a cabo dentro del programa, pero que si eran parte de la implementación, por lo tanto se procederá a explicarlas superficialmente:

1. **ImagenToString:**
   - Permite transformar una imagen a una representación string. Esta debe considerar si el pixel es del tipo PixBit, PixRGB o PixHexadecimal.
2. **DepthLayers:**
   - Separa una imagen en capas según la profundidad de los pixeles. El resultado debe ser una lista de imágenes que agrupa los pixeles que comparten profundidad.
3. **DesCompress:**
   - Método que permite descomprimir una imagen. La lógica que se hubiera utilizado para la creación de este mismo es que dentro del mismo método Compress se guarde la imagen original y entregue una nueva imagen con el pixel comprimido. Al momento de utilizar DesCompress se retornaría la imagen original guardada.

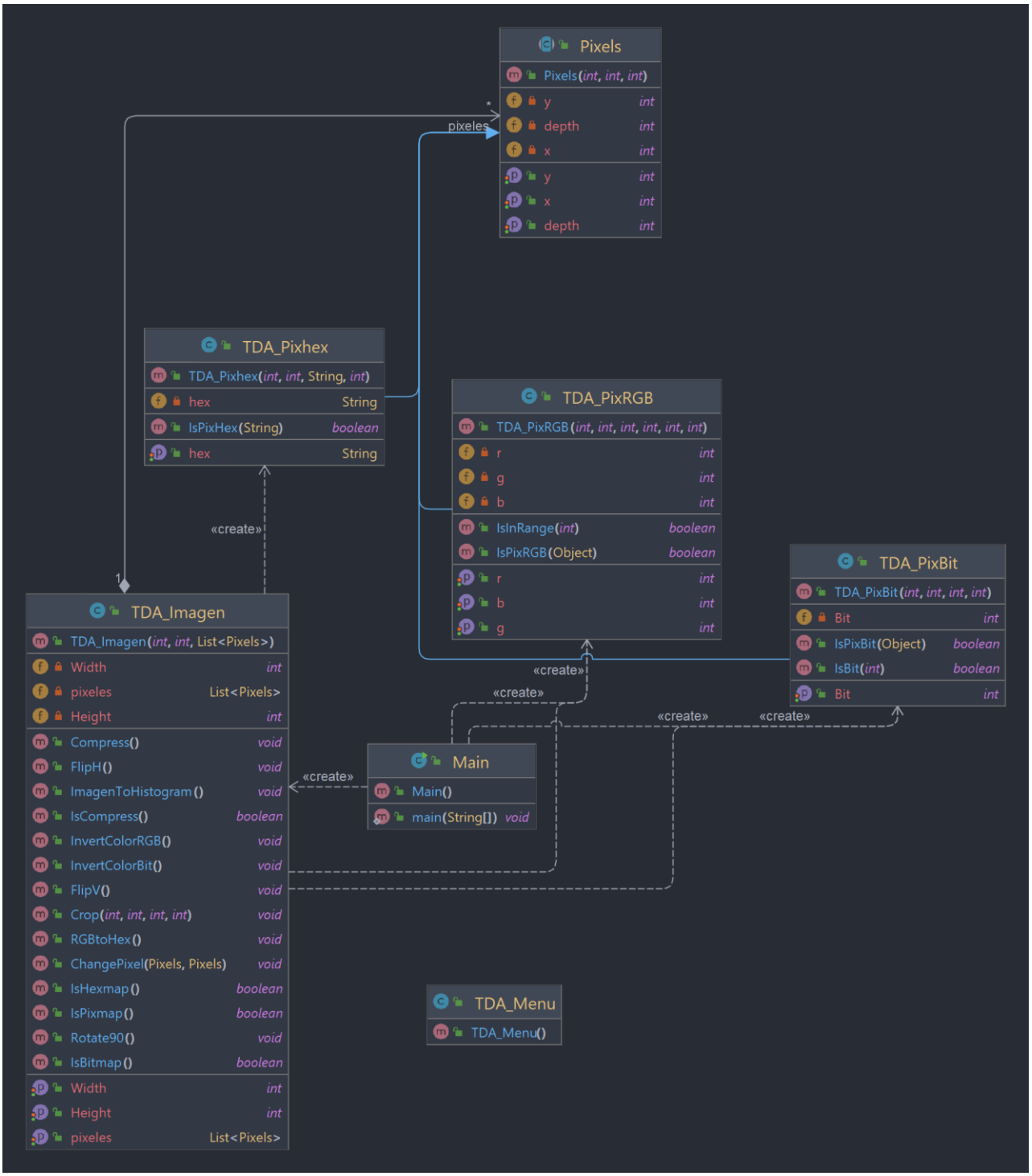
5.5. Diagramas de análisis.

Diagrama creado a partir de intellij. Este era el análisis que originalmente se había previsto.



5.6. Diagrama de diseño.

Diagrama creado a partir de Intellij:



6. Aspectos de la implementación.

6.1. Estructura del proyecto:

El proyecto está estructurado a través de las siguientes clases.

- **TDA Imagen:** Posee el constructor para la aplicación. Tiene sus respectivos getters y setters para su utilización dentro del código
- **Pixeles:** Posee un constructor que almacena las coordenadas X e Y de un pixel, además de su profundidad.
- **TDA PixBit:** Posee el constructor para la aplicación de un pixbit. Su representación es en base a un objeto con los siguientes atributos [Coordenada X, Coordenada Y, Bit, Profundidad/Depth], se debe ser consciente de que las coordenadas y la profundidad son heredados de los pixeles.
- **TDA PixHex:** Posee el constructor para la aplicación de un pixhex. Su representación es en base a un objeto con los siguientes atributos [Coordenada X, Coordenada Y, Color hexadecimal, Depth]. Hereda los mismos atributos que PixBit.
- **TDA PixRGB:** Posee el constructor para la aplicación de un PixRGB. Su representación es en base a un objeto con los siguientes atributos [Coordenada X, Coordenada Y, Color rojo, Color verde, Color azul, Depth]. Hereda lo mismos atributos que PixBit y PixHex.
- **MAIN:** Corresponde al archivo que contiene los ejemplos de uso del programa.

Cabe destacar que el laboratorio no trabaja con archivos externos al main, tampoco se utilizaron librerías especiales, sino que solo se utilizó predicados nativos del lenguaje Java.

Una vez abierto la carpeta SRC, podemos encontrar el código organizado con los siguientes archivos:

- Main
- Pixeles
- Imagen
- Menu
- PixBit
- Pixhex
- PixRGB

Para este proyecto no se considero ninguna biblioteca en particular.

6.2. Compilador o interprete usado:

Para este proyecto es necesario el IDE Intellij SDK – java versión 17.0.4 con un nivel de lenguaje SDK default. En el caso del JDK, se utilizo la versión 11.0.13.8 para compilar el programa.

Se desconoce si es posible correr el programa dentro de otro IDE como Visual Studio Code, Apache, Eclipse o versiones online de JAVA.

7. Instrucciones de uso.

Para el correcto uso del programa sin errores, se debe tener instalado el compilador o interprete usado en las versiones mencionadas.

Se debe llegar dentro de la carpeta SRC el cual contiene un archivo main el cual debe ser ejecutado. Como no se alcanzo a crear una interfaz para la interacción del usuario, este archivo contiene ejemplo de uso.

Para correr el archivo de ejemplo, se debe seguir los siguientes pasos.

1- Abrir la carpeta contenedora de todos los archivos.
   EJ: C:\Users\usuario \Escritorio\Laboratorio3_paradigmas_java\Laboratorio3_xxxxxxx_alumno.
2- Abrir esta dirección en el CMD
3- Utilizar el script de compilación "batch.bat"

Esto debería lograr que el main funcione correctamente y muestre los ejemplos de uso.

8. Resultados.

Requerimientos empleados dentro del programa y su nivel de implementación:

- 100% -> El constructor o predicado se implemento de manera correcta y con los resultados solicitados.
- 50% -> El predicado se implementó y se pensó en su diseño de solución, pero el resultado dado no cumple con el objetivo solicitado.
- 0% -> El predicado directamente no fue implementado. (No se supo implementar)

Método	Grado de alcance.
TDA image - Constructor	100%
TDA pixbit - Constructor	100%
TDA pixhex - Constructor	100%
TDA pixrgb – Constructor	100%
isBitmap – Predicado	100%
Ispixmap - Predicado	100%
isHex - Predicado	100%
flipH – Predicado	100%
flipV – Predicado	100%
Crop – Predicado	100%
imageRGBToHex – Método	100%
imageToHistogram – Método	100%
imageRotate90 – Método	100%
Compress – Método	100%
changePixel – Método	100%
InvertColorBit – Método	100%
invertColorRGB – Método	100%
Iscompressed – Método	100%
imageToString – Método	0%
imageDepthLayers – Método	0%
imageDecompress - Método	0%

Como una autoevaluación, se marcan las casillas de diferentes colores en la parte de grado de alcance, para una visualización más notoria con respecto al éxito o fracaso logrando en cada implementación de los TDA.

9. Conclusiones.

Después de realizar el proyecto, se puede concluir que se cumplió el objetivo principal de este laboratorio que consiste en aprender el lenguaje de programación JAVA y el paradigma orientado objetos de forma correcta, pues a pesar de que algunos métodos no se pudieron implementar, la mayoría se pudo llevar a cabo la solución junto con el aprendizaje que se obtuvo en clases.

Se tuvo bastantes dificultades a la hora de llevar a cabo el proyecto, las mayores dificultades fue saber con que versión de java se estaba trabajando en un inicio, pues al ser un IDE completamente diferente a lo utilizado y un nuevo lenguaje que tiene su estructura, en un principio no se reconoce como utilizar cada cosa. Un ejemplo de esto es que en los comit iniciales el archivo estaba creado con “x.class”, lo cual no correspondía y se tuvo que crear un archivo desde el principio. Otro error cometido debido a esto es trabajar con una versión diferente a la solicitada.

Otra dificultad que se enfrento en este laboratorio fue la falta de tiempo. Pues por algún motivo este laboratorio se sintió con mucho menos tiempo para trabajar en comparación a los 2 anteriores.

Comparando este paradigma con los presentando antes (funcional y lógico), el código del paradigma orientado a objetos resulto mucho más corto y simple, permitiendo implementar incluso más funciones que en los laboratorios pasados. Además, la IDEA intellij permitía saber en qué parte del código había errores y generaba automáticamente los getters y setters.

Se puede concluir que con este laboratorio final del semestre, se logro utilizar el paradigma esperado y utilizar el conocimiento aprendido para posteriores asignaturas, además de su uso trabajando dentro de la industria.