Informe Laboratorio 3
Paradigma Orientado a objetos
Programación en Lenguaje Java

Nombre: Iván Alejandro Guajardo Arias.
Curso: Paradigmas de programación.
Sección: A-1.
Profesor: Gonzalo Martinez.

03 de Noviembre, 2022.

Tabla de contenidos:

2. Introducción

En el informe que se presenta a continuación, se expone el tercer laboratorio del ramo "Paradigmas de programación"; el cual, se basa en el paradigma orientado a objetos, por medio del lenguaje multipropósito, Java. El proyecto consiste en desarrollar un programa que facilite la manipulación de imágenes con un enfoque simplificado, utilizando el, ya mencionado, lenguaje Java, usando el IDE IntelliJ de JetBrains para trabajarlo durante su desarrollo.
Para lograr el objetivo general del laboratorio, se necesita entender el enunciado, esta vez, desde la perspectiva orientada a objetos (a diferencia de las 2 ocasiones anteriores) y, desde el mismo paradigma, observar los requerimientos para llevarlos a los distintos tipos de clases que podrían permitir la manipulación adecuada de las imágenes. Dentro de los pasos consecuentes, se pueden encontrar los siguientes:

1. Entender el propósito del proyecto y abstraer el problema con paradigma orientado a objetos, pues Java permitiría la posterior relación entre todas las clases, de modo que cada método y atributo pueda coexistir adecuadamente para el objetivo que se tiene.
2. Proponer un UML que permita la comprensión de las clases y cómo éstas interactúan dentro del programa, de esta manera, ordenarlas previamente y poder implementar todo en código sin mayores adversidades
3. Utilizar correctamente los atributos de cada clase para poder implementar los métodos y aprovechar las facilidades que la herencia entrega; de esta manera, armonizar el código en la medida de lo posible.

En lo que viene de informe, se verá la descripción del paradigma, se hará un marco teórico que facilite la comprensión del problema y, facilite así, el análisis de este. De esta manera, se podrá comprender además cómo el paradigma orientado a objetos puede facilitar la resolución de problemas de manera efectiva desde una perspectiva distinta, abstrayendo todo como la interacción entre distintos objetos que permiten llevar a cabo los procesos de manera efectiva, por medio de encapsulamiento y herencia.

3. Descripción del paradigma

Empezaremos por definir el paradigma orientado a objetos. ¿De qué se trataría este caso? Según lo aprendido en clases, se entiende que el paradigma orientado a objetos consiste en un paradigma que plantea todos los conceptos como objetos; sin embargo, ante la generalización de estos se utilizan las clases, puesto que los objetos serán las instancias de éstas.

Las clases consisten, como ya mencionamos, en la generalización de los objetos, y proponen el cómo sería cada "tipo", por medio de atributos (las cualidades que cada objeto de la clase podría tener) y métodos (parafraseando un poco lo presentado en clases: serían comportamientos que podrían tener los objetos, siendo similares a las funciones, vistas en el paradigma funcional)

"Este tipo de programación se emplea para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos. " (Darias Pérez)

Así pues, todo en este paradigma dependerá de dichas relaciones "piezas" mencionadas, reutilizando código y haciendo que todo coexista, muy similar a cómo percibimos la realidad normalmente.

3.1 Marco teórico

Lo más básico para poder resolver el problema, sería empezar por el proceso de abstracción. Dicho proceso se tiende a considerar como el traducir un problema a la generalidad, llevándolo a distintos niveles y subprocesos que faciliten el desarrollo de este. No obstante, esta vez se lleva a un nuevo paradigma, en el cual, deben ser definidas las distintas piezas que coexisten en el programa y, así, permiten a cada método realizarse.

Ahora, sería prudente si también se menciona lo que serían los subprocesos. En el caso del paradigma orientado a objetos, contempla múltiples herramientas que facilitan las soluciones de distintas maneras; en este caso, se consideró el uso de una interfaz, composición y agregación para llevar las herencias entre objetos a cabo.

Por otro lado, también es importante describir el trabajo en que se basa el lenguaje Java. Así pues, éste es una de las 2 opciones contempladas (junto a C#) para la realización de este laboratorio, y fue elegido principalmente porque la mayoría de la materia vista en clases ha sido demostrada con este lenguaje. Teniendo en cuenta, se comenta además que Java funciona muy bien en IntelliJ y se entregan herramientas muy versátiles para la escritura y testeo del código, obteniendo más facilidades que en cualquier otro laboratorio rendido en el ramo.

4. Problemática

4.1 Descripción del problema

El problema que el enunciado propone, consiste en el desarrollo de un software que permita la manipulación de imágenes, así como photoshop o GIMP. Sin embargo, este tiene un enfoque simplificado de lo que serían dichos programas, pero permitiría igualmente el uso de funciones tales como la rotación de imagen, voltearla, transformarla, editarla, etc. Otra diferencia destacable entre los softwares mencionados y el propuesto por el laboratorio, es que este último tendría un enfoque principalmente en imágenes RGB/RGB-D y representaciones sencillas de las imágenes, como listas de píxeles u otras.

4.2 Análisis del problema

A grandes rasgos, el problema consiste en presentar un software que permita la edición de imágenes, como los ya señalados. Entonces, lo primero sería considerar que las imágenes consisten en arreglos bidimensionales de píxeles, por lo que sería importante disponer de funciones que permitan la manipulación de los píxeles, sus valores, sus posiciones, etc. Considerando la perspectiva de este último laboratorio, es que se deja de lado el uso de listas como estructura estelar para poder contener la información de cada píxel y, en su lugar, se ocupan las clases y sus atributos para alojar la información relevante, aunque algunos de estos atributos han sido igualmente listas. Por otro lado, también se consideran los requerimientos solicitados para el laboratorio, los cuales, se pueden observar en la tabla de autoevaluación, anexada al final del informe (ver tabla 3).

A partir de lo dicho anteriormente, es que se puede intuir que los constructores consisten en métodos que permiten instanciar un objeto, dándole un valor adecuado a cada atributo, dependiendo del tipo de píxel dado o cualquier otro objeto posible que se solicite mediante el menú.

Otro aspecto que ha cambiado bastante el panorama, ha sido que se implementa una opción de menú para poder interactuar con el programa, por lo que todo el laboratorio debe tener una leve orientación a poder ser interactivo para el usuario final y, consecuentemente, crear instancias en las que se permita la manipulación de objetos en distintas situaciones a las formuladas anteriormente.

4.3 Otras consideraciones

Se pueden ingresar **N** píxeles a cada imagen, considerando su ancho Ancho y su alto Largo; esto implica entonces la posibilidad de una imagen infinitamente grande, o una diminuta. A partir de ello, es que se podrían considerar los valores Ancho y Largo para poder implementar funciones de volteo, o la posibilidad de un TDA para todo tipo de pixel, puesto que los atributos permiten la manipulación de la información de una manera que no se había experimentado previamente en los otros laboratorios. Así pues, existe la posibilidad de separar los casos a trabajar sin la necesidad de crear más subclases.

De esta manera, se lograría finalmente crear imágenes en base a listas de píxeles, modificarlas en base a modificar los valores de los píxeles, convertirlas, preguntar de qué tipo son, etc.

5. Diseño de solución

5.1 Descripción de la solución

La solución propuesta, plantea que en un inicio se trabaje desde lo más básico hasta lo más complejo, empezando por píxeles de cada tipo, sus funciones generales y, desde ahí, retomar con lo que corresponde a imágenes como tal, que es lo que aparece en el enunciado del laboratorio. De esta manera, se definen los TDAs Pixel, Color, Par, Menú y el de imagen.
Por su parte, se podría considerar cada TDA como una clase con sus respectivos atributos y métodos. Finalmente, por medio de herencia, se pueden implementar métodos que faciliten la perspectiva del qué se quiere obtener, proyectando un resultado que cumpla con lo solicitado en cada aspecto de la evaluación.
En Píxel, se considera posición x, posición y, y su color que le da un significado como píxel, más allá de su ubicación en la imagen y el tipo de píxel; llamando por ejemplo a Pixel(0, 0, 5, Color) (x, y, profundidad, Color). Posteriormente, se consideran también los métodos que facilitan la manipulación de los Pixeles a usar en las imágenes, por ejemplo, sus setters.

No se considera como la solución más refinada, puesto que se ha ido mejorando el entendimiento del paradigma en la medida que se ha ido avanzando con el laboratorio. Siguiendo la misma línea, se puede mencionar, por ejemplo, que se pudo haber utilizado una clase abstracta que permita organizar de mejor manera los datos generales de cada tipo de pixel y, por medio de herencia, implementar los distintos tipos de pixeles a utilizar. También se han ido descubriendo distintas herramientas de Java que, de haber sido utilizadas, habrían ahorrado mucho código y posibles bugs; sin embargo, en honor al periodo, no fueron consideradas finalmente.

5.2 TDAs

Los pixeles consisten en una lista que considera sus posiciones x e y, su color y la profundidad de este. De esta forma, se consideran los ya mencionados como los siguientes ejemplos: (ver tabla 1 en Anexos)

Por otro lado, en experiencias anteriores se observó que estos pixeles consideran situaciones similares y, por ende, conductas que se replican en cada uno. De este modo, gracias a la posibilidad entregada por el paradigma, de poder definir atributos, es posible plantear todos estos métodos en una única clase que diferencie cada caso, de acuerdo al color asignado al momento de instanciar el Pixel.

El TDA Color fue implementado en este laboratorio, dado que facilita en gran medida la manipulación de cada Pixel, puesto que es posible crear distintos tipos, de acuerdo a lo solicitado, dando por ejemplo la posibilidad de trabajarlos como bits, hexs o rgbs por medio de la sobrecarga del constructor e instanciando en cada caso los atributos con un valor por defecto que no será leído en caso de no ser del tipo indicado.

El TDA Par se utiliza para poder implementar el histograma de la imagen, puesto que se quiere tratar una lista con valor y frecuencia, siendo que el primer valor podría tomar tipo String o int, por lo que se opta por manipular el valor convertido en String.

El TDA Imagen pasa primero por la interfaz imagen, que consiste en una serie de métodos mandatorios a implementar en la Imagen finalmente. A continuación de eso, por medio de herencia, se implementa Imagen, que consiste en un Ancho, un Largo y una lista de tipo Pixel para manipular; por medio de la composición, se logra dar forma tanto a los Pixeles como a las Imágenes y la herencia facilita el resto del proceso.

Finalmente, el TDA Menu consiste en las distintas interacciones que el programa puede ofrecer al usuario, teniendo como atributos listas de imágenes, de pixeles y de colores, para así integrar las 3 clases más importantes y operar con ellas sin borrarlas después de cada uso.

(ver diagrama UML en Anexos)

TDAs implementados:

a. Image
b. Pixel
c. Par
d. Color
e. Menu

5.3 UMLs

Durante el proceso de diseño, se creó un UML para poder ordenar las ideas detrás de las clases, sin embargo, desde el primero hasta el último, las ideas fueron cambiando constantemente, tanto a nivel de conexión como de implementación. Así pues, se obtienen los 3 UML anexados al final.

6. Aspectos de implementación

La estructura a seguir por el proyecto, contempla el uso de subclases que faciliten el trabajo realizado por cada método solicitado. De esta manera, se contempla un archivo main (para la ejecución del programa), las 5 clases descritas por los TDA y la interfaz de Image (llamada Inter-Image)

Otro aspecto relevante, sería la prohibición de cualquier uso de programación bajo otro paradigma, por lo que todo el laboratorio está enfocado netamente al uso de los TDA mediante representaciones tales como objetos. Además de mencionar las siguientes especificaciones utilizadas por el entorno: || Build system: Gradle || JDK: Azul Zulu versión 11.0.17.|| Versión de Java: 11.0.16.1 || Windows 10

7. Instrucciones de uso

Pre-requisitos: Para el uso del programa es necesario tener la versión de Java 11.0.16.1 y Gradle 7.6, junto a sus respectivas rutas para poder ejecutar sus comandos desde la consola.

Cómo iniciar el programa:

1. Una vez instalados los pre-requisitos, basta con dirigirse a la carpeta "main" dentro del proyecto.
2. Estando ahí, hacer doble click en el archivo **"main.bat"** (se ejecutará gradle y, a continuación, el proyecto desde una ventana cmd).
3. En la nueva ventana, se ejecuta el laboratorio desde su menú.

Estando ya en el programa:

1. **Primero, se presentará el menú para la interacción con el usuario, entregando 4 opciones (crear imagen, modificar imagen, visualizar imagen, salir)**
2. **Se ingresa una opción válida de acuerdo a los números correlativos a cada una y se presiona "Enter" (en adelante, los valores se ingresan siempre de la misma manera).**
3. **En el menú de crear, se preguntará por el largo y ancho de la imagen (de acuerdo a lo mencionado recién), luego se solicitará del tipo de píxel para cada uno (no está permitido cambiar el tipo de píxel durante la creación de una imagen), y luego se solicitarán los valores correspondientes a cada uno (color y profundidad, respetar los valores señalados. En caso de ser hex, adecuarse a lo que un hex puede ser, o sea, valores entre el 0 y el 9, y caracteres entre A y F; en total, 6). Tras crear la imagen, se devuelve al menú principal.**
4. **Menú de modificaciones: Primero, se mostrará un mensaje que señala cuántas imágenes existen en el programa actualmente, solicitando que se elija alguna (elegir el número correspondiente. En caso de ser la primera, es la número 1 y así sucesivamente). A continuación, se presentan las opciones de modificación (Voltear horizontalmente, voltear verticalmente, crop, rgb to hex, rotate90 o volver al menú principal). Tras elegir la opción, se mostrará la imagen modificada, se agrega a la cola de la lista de imágenes y se volverá al menú principal.**
5. **En el menú de visualización, preguntará qué imagen se desea visualizar, se dispondrá de 2 opciones (ver la imagen como tal o como histograma). Tras elegir la opción, se visualizará la imagen en el formato indicado.**
6. **En caso de elegir la opción de "Salir" (4), el programa se despide y se cierra la ventana.**

8. Resultados

8.1 Ideales

Idealmente, se espera un software de uso simple que permita trabajar píxeles representados como listas. Se logra de manera exitosa el realizar las siguientes acciones en píxeles e imágenes modificar, convertir, manipular, etc. Además, se hace correcto uso del proceso de abstracción para proponer clases que permitan implementar cada método y así logren relacionar los objetos instanciados por el usuario mediante el menú, instanciando objetos y modificándolos (lo principal del paradigma es la comprensión de la realidad mediante el uso de objetos).

8.2 Reales

Se lograron implementar los primeros 13 requerimientos funcionales del enunciado, hasta rotate90. Esto demuestra un buen aprendizaje a nivel estudiante, ya que se usó provechosamente el tiempo y se dio un correcto uso a los TDA, herencia, encapsulación, paso de mensajes y otros aspectos vistos en clase. Satisfactoriamente, funciona cada función como debería, excepto por el histograma, que no permite eliminar elementos repetidos y, por lo mismo, los muestra tantas veces como aparezcan (a pesar de mostrar valores y frecuencias como corresponde).

8.3 Posibles fallos en el estudiante y análisis de lo no logrado

Principalmente, se destaca la falta de orden y claridad con conceptos básicos del paradigma en un inicio, los cuales, se fueron puliendo en la medida que se iba avanzando con el proyecto, además de la dificultad inicial para aprender Java. De realizar el laboratorio nuevamente, se plantearía un diagrama UML distinto al propuesto, se utilizaría una clase abstracta y la interacción con el menú se propondría de otro modo.

De los requisitos no implementados, están: compress, changePixel, invertColorBit, invertColorRGB, image to string, depthLayers y decompress. Lamentablemente, no se alcanzaron a realizar, sin embargo, se observa que se podrían implementar si se considera por ejemplo, que para invertir el color de un bit basta con cambiar de 1 a 0 y viceversa. Gracias al paradigma orientado a objetos, se puede concluir que se trabajan de manera similar a métodos conocidos durante esta experiencia.

8.4 Autoevaluación

En aspectos generales, se considera un buen desempeño en cuanto al estudiante. A pesar de ello, vale la pena mencionar que existe un desorden en sus TDAs, por lo que podría considerarse un éxito. Ver tabla 3, adjunta en Anexos.

9. Conclusión

Dentro de todo, se considera como un éxito a la segunda experiencia realizada en el laboratorio de paradigmas, puesto que se cumplió con el mínimo de requerimientos solicitados satisfactoriamente (casi todos los que se implementaron). El menú funciona adecuadamente y se pueden aplicar todos los procesos solicitados desde las interacciones. La mayor limitación en este caso, fue nuevamente, la falta de claridad que tuvo el estudiante desde un inicio pues, en esta ocasión, existían algunos conceptos que le costó asimilar en el nuevo paradigma, principalmente. Después de algunos días de demora, el estudiante logra establecer un avance significativo y esto le permite concluir lo suficiente; de todos modos, es importante mejorar las buenas prácticas que el estudiante posee para las siguientes entregas, de modo que se optimice el trabajo y se puedan cumplir mejor los plazos, a la vez que se obtenga mayor legibilidad para el código.

Sobre el paradigma, el estudiante concluye que es una propuesta bastante interesante que permite abstraer procesos de una manera similar a la cotidiana y, además, se hace una pequeña relación con lo visto en estructuras de datos. Recuerda un poco a la noción de structs y entrega una alternativa altamente versátil y flexible que permite resolver distintas problemáticas de muchas maneras. La reflexión de este laboratorio apunta al cómo el paradigma facilita el desarrollo de aplicaciones mediante el usto de objetos, y motiva al estudiante a seguir desarrollándose en este ámbito. Sobre las restricciones, se puede mencionar que el paradigma, debido a su alta flexibilidad, permite una libertad tremenda que, de no tener orden, puede agobiar a quien programe, por lo que es particularmente importante realizar un proceso adecuado de abstracción.

10. Anexos

Tabla 1

TDA	Plantilla	Ejemplos
Color	(bit | hex | r g b)	(0, 1, (0|1), 4) | (1, 1, #123456)
Pixel	(pos_x pos_y depth color)	(0, 1, 4, Color de arriba)
Par	(String value, long frequency)	(“1”, 5)

Tabla 2

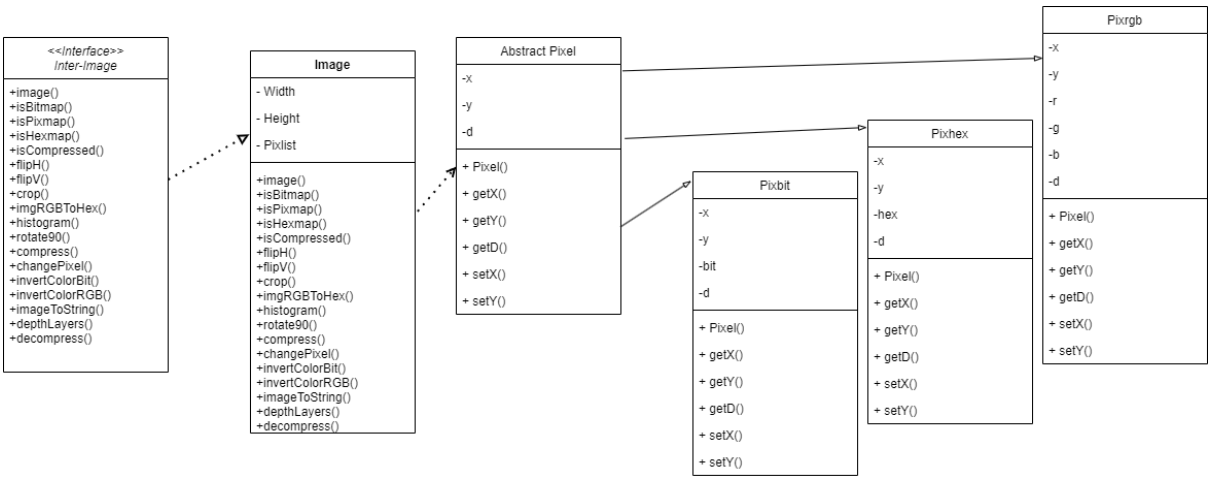
image	image(Ancho, Largo, Pixlist).	(2 2 pixlist1)

Tabla 3

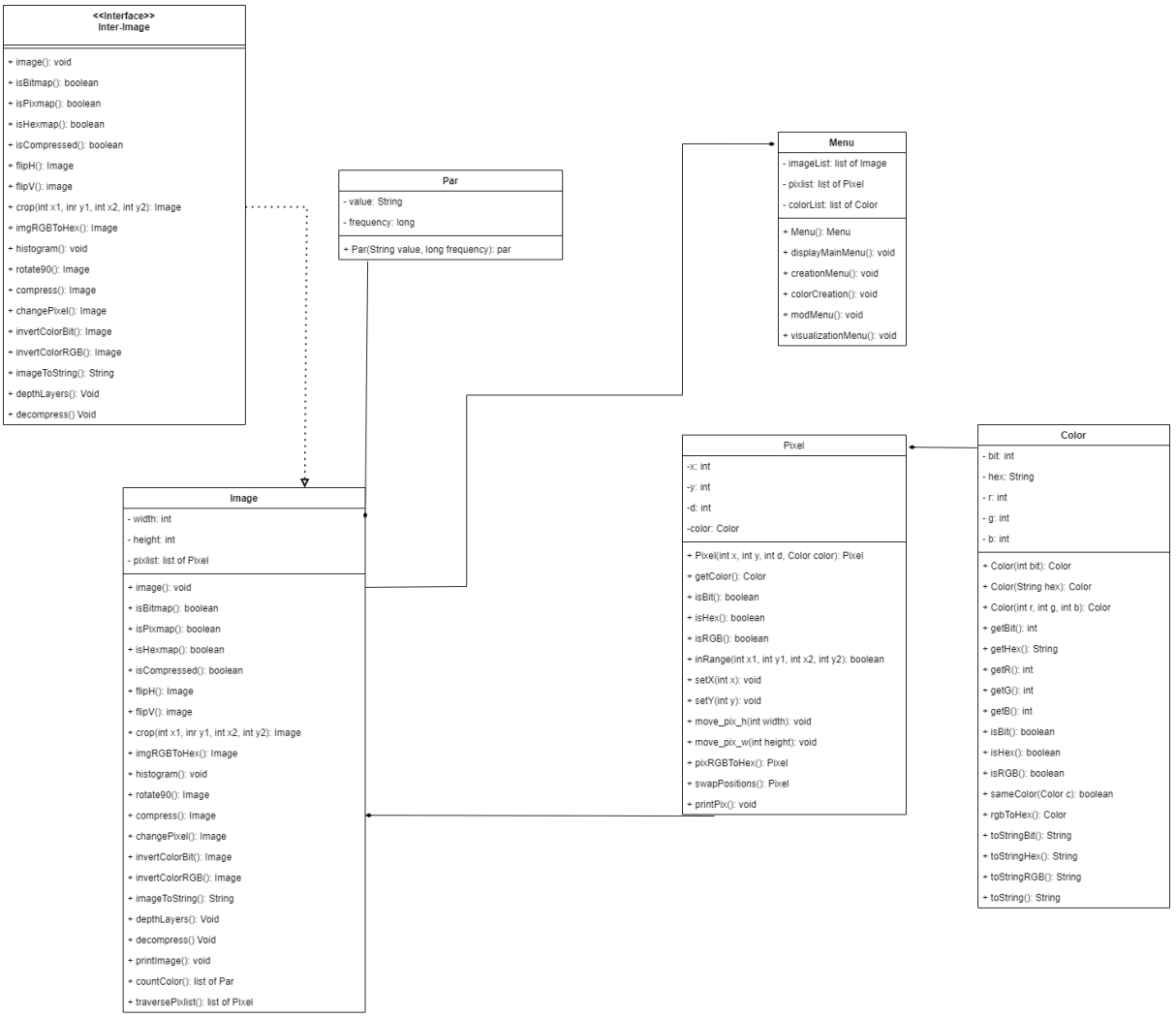
TDA	Valor (logrado | parcialmente logrado | no logrado)
Clases y estructuras	**logrado 1**
TDA image - constructor	**logrado 1**
TDA image - bitmap?	**logrado 1**
TDA image - pixmap?	**logrado 1**
TDA image - hexmap?	**logrado 1**
TDA image - compressed?	**logrado 1**
TDA image - flipH	**logrado 1**
TDA image - flipV	**logrado 1**
TDA image - crop	**logrado 1**
TDA image - imgRGB->imgHex	**logrado 1**
TDA image - histogram	**parcialmente logrado 0.5**
TDA image - rotate90	**logrado 1**
TDA image - compress	**no logrado 0**

TDA image - changePixel	**no logrado 0**
TDA image - invertColorBit	**no logrado 0**
TDA image - invertColorRGB	**no logrado 0**
TDA image - image->string	**no logrado 0**
TDA image - depthLayers	**no logrado 0**
TDA image - decompress	**no logrado 0**

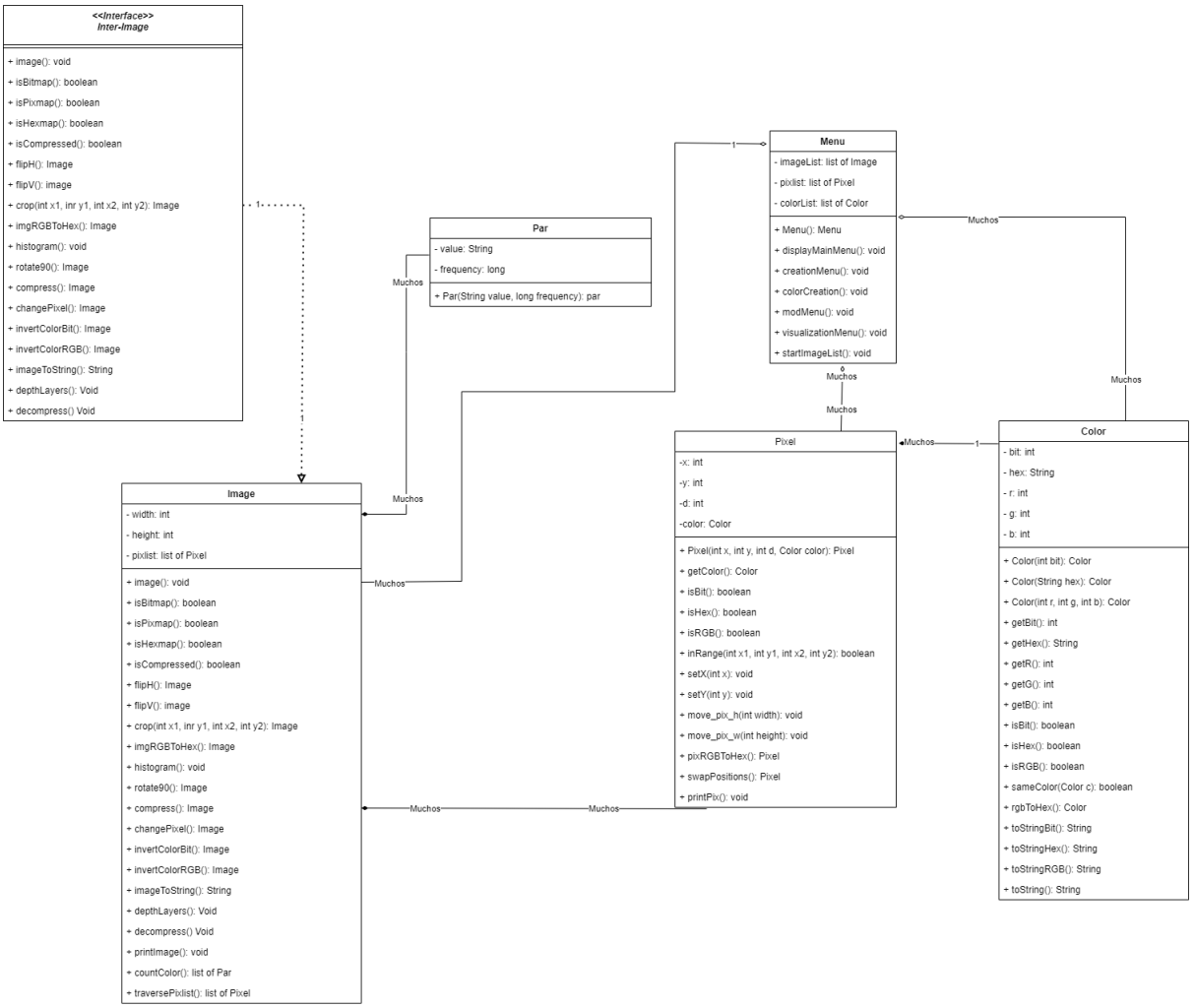
PRIMER UML



SEGUNDO UML


<<Interface>>
Inter-Image
+ image(): void
+ isBitmap(): boolean
+ isPixmap(): boolean
+ isHexmap(): boolean
+ isCompressed(): boolean
+ flipH(): Image
+ flipV(): image
+ crop(int x1, inr y1, int x2, int y2): Image
+ imgRGBToHex(): Image
+ histogram(): void
+ rotate90(): Image
+ compress(): Image
+ changePixel(): Image
+ invertColorBit(): Image
+ invertColorRGB(): Image
+ imageToString(): String
+ depthLayers(): Void
+ decompress() Void
Par
- value: String
- frequency: long
+ Par(String value, long frequency): par
Menu
- imageList: list of Image
- pixlist: list of Pixel
- colorList: list of Color
+ Menu(): Menu
+ displayMainMenu(): void
+ creationMenu(): void
+ colorCreation(): void
+ modMenu(): void
+ visualizationMenu(): void
Pixel
-x: int
-y: int
-d: int
-color: Color
+ Pixel(int x, int y, int d, Color color): Pixel
+ getColor(): Color
+ isBit(): boolean
+ isHex(): boolean
+ isRGB(): boolean
+ inRange(int x1, int y1, int x2, int y2): boolean
+ setX(int x): void
+ setY(int y): void
+ move_pix_h(int width): void
+ move_pix_w(int height): void
+ pixRGBToHex(): Pixel
+ swapPositions(): Pixel
+ printPix(): void
Color
- bit: int
- hex: String
- r: int
- g: int
- b: int
+ Color(int bit): Color
+ Color(String hex): Color
+ Color(int r, int g, int b): Color
+ getBit(): int
+ getHex(): String
+ getR(): int
+ getG(): int
+ getB(): int
+ isBit(): boolean
+ isHex(): boolean
+ isRGB(): boolean
+ sameColor(Color c): boolean
+ rgbToHex(): Color
+ toStringBit(): String
+ toStringHex(): String
+ toStringRGB(): String
+ toString(): String
Image
- width: int
- height: int
- pixlist: list of Pixel
+ image(): void
+ isBitmap(): boolean
+ isPixmap(): boolean
+ isHexmap(): boolean
+ isCompressed(): boolean
+ flipH(): Image
+ flipV(): image
+ crop(int x1, inr y1, int x2, int y2): Image
+ imgRGBToHex(): Image
+ histogram(): void
+ rotate90(): Image
+ compress(): Image
+ changePixel(): Image
+ invertColorBit(): Image
+ invertColorRGB(): Image
+ imageToString(): String
+ depthLayers(): Void
+ decompress() Void
+ printImage(): void
+ countColor(): list of Par
+ traversePixlist(): list of Pixel

UML FINAL


<<Interface>>
Inter-Image
+ image(): void
+ isBitmap(): boolean
+ isPixmap(): boolean
+ isHexmap(): boolean
+ isCompressed(): boolean
+ flipH(): Image
+ flipV(): image
+ crop(int x1, inr y1, int x2, int y2): Image
+ imgRGBToHex(): Image
+ histogram(): void
+ rotate90(): Image
+ compress(): Image
+ changePixel(): Image
+ invertColorBit(): Image
+ invertColorRGB(): Image
+ imageToString(): String
+ depthLayers(): Void
+ decompress() Void
Par
- value: String
- frequency: long
+ Par(String value, long frequency): par
Menu
- imageList: list of Image
- pixlist: list of Pixel
- colorList: list of Color
+ Menu(): Menu
+ displayMainMenu(): void
+ creationMenu(): void
+ colorCreation(): void
+ modMenu(): void
+ visualizationMenu(): void
+ startImageList(): void
Image
- width: int
- height: int
- pixlist: list of Pixel
+ image(): void
+ isBitmap(): boolean
+ isPixmap(): boolean
+ isHexmap(): boolean
+ isCompressed(): boolean
+ flipH(): Image
+ flipV(): image
+ crop(int x1, inr y1, int x2, int y2): Image
+ imgRGBToHex(): Image
+ histogram(): void
+ rotate90(): Image
+ compress(): Image
+ changePixel(): Image
+ invertColorBit(): Image
+ invertColorRGB(): Image
+ imageToString(): String
+ depthLayers(): Void
+ decompress() Void
+ printImage(): void
+ countColor(): list of Par
+ traversePixlist(): list of Pixel
Pixel
-x: int
-y: int
-d: int
-color: Color
+ Pixel(int x, int y, int d, Color color): Pixel
+ getColor(): Color
+ isBit(): boolean
+ isHex(): boolean
+ isRGB(): boolean
+ inRange(int x1, int y1, int x2, int y2): boolean
+ setX(int x): void
+ setY(int y): void
+ move_pix_h(int width): void
+ move_pix_w(int height): void
+ pixRGBToHex(): Pixel
+ swapPositions(): Pixel
+ printPix(): void
Color
- bit: int
- hex: String
- r: int
- g: int
- b: int
+ Color(int bit): Color
+ Color(String hex): Color
+ Color(int r, int g, int b): Color
+ getBit(): int
+ getHex(): String
+ getR(): int
+ getG(): int
+ getB(): int
+ isBit(): boolean
+ isHex(): boolean
+ isRGB(): boolean
+ sameColor(Color c): boolean
+ rgbToHex(): Color
+ toStringBit(): String
+ toStringHex(): String
+ toStringRGB(): String
+ toString(): String
1
Muchos

11. Bibliografía

a. https://www.youtube.com/watch?v=uudsNL2G7nE&list=PLGfF3KgbxaiwLDxZaSuec2zxNZC7zmQkI

b. https://stackoverflow.com
c. https://www.cosy.sbg.ac.at/~pmeerw/Watermarking/lena.html
d. https://uvirtual.usach.cl/moodle/

e. Profesor Gonzalo Martinez.
f. https://www.w3schools.com/java/java_intro.asp
g. https://intelequia.com/blog/post/3072/qué-es-la-programación-orientada-a-objetos#:~:text=La%20Programación%20Orientada%20a%20Objetos%20(POO)%20es%20un%20paradigma%20de,concepto%20de%20clases%20y%20objetos.