**INFORME DE LABORATORIO 3: SIMULACIÓN DE UN PROGRAMA PARA REALIZAR UN TRATAMIENTO DE IMÁGENES SIMPLIFICADO**

Nombre: Aracely Castro V.

Profesor: Roberto Gonzales I.

Asignatura: Paradigmas de Programación (2/2022)

Índice

[1. INTRODUCCIÓN 2](#_Toc119155547)

[**1.1** **DESCRIPCIÓN DEL PROBLEMA** 2](#_Toc119155548)

[**1.2** **DESCRIPCIÓN DEL PARADIGMA** 3](#_Toc119155549)

[2. DESARROLLO 3](#_Toc119155550)

[**2.1** **ANÁLISIS DEL PROBLEMA** 3](#_Toc119155551)

[**2.2** **DISEÑO DE LA SOLUCIÓN** 3](#_Toc119155552)

[**2.3** **ASPECTOS DE IMPLEMENTACIÓN** 3](#_Toc119155553)

[**2.3.1** **EJEMPLOS DE USO** 3](#_Toc119155554)

[**2.3.2** **RESULTADOS ESPERADOS** 3](#_Toc119155555)

[**2.3.3** **POSIBLES ERRORES** 3](#_Toc119155556)

[**2.4** **RESULTADOS Y AUTOEVALUACIÓN** 3](#_Toc119155557)

[**2.4.1** **RESULTADOS** 3](#_Toc119155558)

[**2.4.2 AUTOEVALUACIÓN** 4](#_Toc119155559)

[3. CONCLUSIÓN 4](#_Toc119155560)

[4. BIBLIOGRAFÍA Y REFERENCIAS 4](#_Toc119155561)

[5. ANEXOS 4](#_Toc119155562)

# **INTRODUCCIÓN**

Con el objetivo de aplicar conceptos del paradigma de programación orientado a objetos usando el lenguaje de programación Java para el desarrollo de un tratamiento de imágenes simple cuyas interacciones se puedan lograr por consola cmd, a continuación, se dará detalle sobre el proceso de solución al problema implementado a través del IDE Apache NetBeans versión 15. El informe constará de una breve introducción de cómo surgió el problema, una descripción del paradigma utilizado, el análisis del problema y como fue el diseño de las soluciones para algunos métodos, sus aspectos de implementación, las instrucciones necesarias para compilar el archivo con el script.bat junto con ejemplos de algunas interacciones del programa, los resultados y la autoevaluación. Finalmente, se dará una conclusión respecto a todo lo anterior.

## **DESCRIPCIÓN DEL PROBLEMA**

Al igual que los laboratorios anteriores se busca desarrollar la simulación de un programa para el tratamiento de imágenes de forma simplificada como GIMP y Adobe Photoshop con la diferencia de que ahora se podrá interactuar a través de la consola. Este simulador permitiría crear imágenes con distintos formatos de píxeles sobre los cuales se pueden aplicar distintas operaciones como, por ejemplo, recortar una imagen, invertir una imagen, obtener histograma de la imagen. Para implementar esto, se debe de tener en cuenta que:

Una imagen pixmap-d es una imagen donde cada uno de sus píxeles contiene información de espacio de colores y profundidad (R)ed, (G)reen, (B)lue y (D)epth. Cada color cubre valores entre 0 y 255 formando en conjunto un color del espectro RGB. La profundidad, en cambio, ofrece información más detallada de la imagen como espacio tridimensional.

Una imagen bitmap-d es una imagen donde sus píxeles tiene asociado su color por un bit, negro como 0 y blanco como 1. Cada píxel también tendría una profundidad asociada.

Una imagen hexmap-d es similar a una imagen pixmap-d con la diferencia que los valores RGB son representados en hexadecimal con formato “#RRGGBB”.

## **DESCRIPCIÓN DEL PARADIGMA**

El Paradigma Orientado a Objetos (POO) se basa en la definición de objetos, serían abstracciones de la realidad en donde se pueden extraer características y comportamientos. Un objeto es definido en una clase donde sus características y comportamientos pueden variar. El POO tiene algunos conceptos bastante importantes a la hora de hacer el análisis del diseño de la solución y construir el código, como son:

* **Objetos:** Son instancias de una clase, es decir, representaciones activas en memoria de una clase durante la ejecución de un programa.
* **Clases:** Son la definición de los atributos y métodos de un objeto. Corresponde a la implementación de un TDA.
* **Interfaz:** Solo definen los métodos de una clase que la implemente. Corresponden a la definición del TDA.
* **Atributos:** Son las características de los objetos e indican el estado del objeto, pueden ser cualquier tipo de dato primitivo, TDA u otra clase.
* **Métodos**: Son los comportamientos de los objetos y el equivalente a las funciones en el paradigma imperativo funcional. Los métodos pueden leer o escribir en los atributos del objeto o interactuar con otros.
* **Constructor:** Tipo de método especial para crear objetos con base a una clase. Se encarga también de reservar la memoria necesaria los datos que definen al objeto.
* **Diagrama Lenguaje Unificado de Modelado (UML):** Representación gráfica de las relaciones entre los objetos y componentes que forman la solución con POO. Durante el análisis de la solución solo se utilizó el Diagrama de Clases.
* **Diagrama de clase:** Representación que permite determinar las clases que componen la solución y como se relaciones entre sí, indicando atributos y métodos de cada clase.

# **DESARROLLO**

## **ANÁLISIS DEL PROBLEMA**

Se identifica que para hacer los métodos para el tratamiento de imágenes simples en Java utilizando POO, antes hay que implementar las siguientes clases:

* Image: Objeto con atributos de ancho, largo (int) y lista de pixeles (Object).
* Píxel: Objeto con clase abstracta con atributos coordX, coordY y profundidad (int).
* Pixrgb: Objeto hijo de Píxel con atributos colorR, colorG y colorB (int)
* Pixrgb\_comprimido: Objeto hijo de Píxel con atributos colorRGuardado, colorGGuardado y colorBGuardado (int).
* Pixbit: Objeto hijo de Píxel con atributos bit (int).
* Pixbit\_comprimido: Objeto hijo de Píxel con atributos bitGuardado (int).
* Pixhex: Objeto hijo de Píxel con atributos hex (String).
* Pixhex\_comprimido: Objeto hijo de Píxel con atributos hexGuardado (String).

Y para implementar la ejecución por consola:

* Menu: Objeto con atributos salirMenu (boolean) y una imagen (Image)
* Main: Objeto con método constructo main que ejecuta el programa

Todo lo anterior para poder interactuar aplicando a una imagen los siguientes métodos de Image:

* isBitmap, isHexmap, isPixmap, isCompress (Pertenencia): Verifica si la imagen es bitmap, hexmap, pixmap o si fue comprimida respectivamente.
* flipH, flipV, rotate90 (Modificador): invierte los pixeles de una imagen horizontal, vertical y gira la imagen 90° a la derecha respectivamente
* crop (Modificador): Recorta una imagen a partir de un cuadrante definido por cuatro puntos convirtiéndolo en una nueva imagen.
* imgRGBToHex (Modificador): Convierte una imagen pixmap-d a hexmap-d.
* histogram (Otros predicados): Muestra un histograma de una imagen.
* compress (Modificador): Comprime el color más frecuente de una imagen.
* changePixel (Modificador): Reemplaza un píxel de una imagen por otro nuevo.
* InvertColorBit, InvertColorRGB (Modificador): Invierte el color de un bitmap-d y pixmap-d respectivamente
* imageString (Otros predicados): Entrega una cadena string de la imagen.
* depthLayers (Otros predicados): Entrega una lista de imágenes separados por profundidad.
* decompress (Modificador): Descomprime una imagen comprimida.

Se pide que cada clase tenga su Interfaz (TDA) desarrollado se implemente con lo necesario para cumplir con los requerimientos pedidos y que se pueda probar los métodos por consola.

## **DISEÑO DE LA SOLUCIÓN**

Por otro lado, se distinguen algunos casos particulares para soluciones de algunos requerimientos utilizando el IDE Apache NetBeans como lo son:

Método histogram: Para poder crear el histograma se desarrollaron métodos auxiliares para lograr hacer los pasos de extraer color, contar color y eliminar el color de la lista de píxeles de la imagen. Con base a lo anterior se fue formando la lista de sublistas, donde cada sublista es de solo enteros, siendo los primeros los valores de los colores y el último valor de la lista la cantidad asociada. El ciclo para cuando la lista de píxeles este vacía. Ver la Figura N°1, Figura N°2 y Figura N°3 en ANEXOS para poder ver la salida de histogram con imágenes pixmap-d, bitmap-d y hexmap-d. P.8.

Método imgRGBToString: Para poder crear este método se utilizo un método auxiliar que regresa el equivalente a un número a string hexadecimal, y luego para cada color RGB se obtuvo la parte entera y resto de cada valor dividido en 16 para transformarlo a string con el método anterior. Por medio de concatenaciones se crea el equivalente a string del color RGB para pasarlo como argumento al constructor de Pixhex, que serían los nuevos píxeles de la imagen. Ver la Figura N°X en ANEXOS para observar como es la transformación de colorRGB a string hexadecimal.

Método crop: Para poder crear el crop se instancio un ArrayList<Object> para contener a aquellos pixeles de la imagen cuyas coordenadas x e y estuvieran dentro del rango definido por los argumentos de entradas del método. Luego de “filtrar” los píxeles se vio la necesidad de crear un método auxiliar que cambiara las coordenadas de los nuevos pixeles para que comenzarán desde las coordenadas (0,0) a las coordenadas (ancho-1, largo-1) para así tener una imagen correctamente creada. Ver figura N°Y en ANEXOS para observar como es aplicar crop a una imagen…..

## **ASPECTOS DE IMPLEMENTACIÓN**

Para este proyecto es necesario utilizar algún IDE de Java, como, por ejemplo, Netbeans o inteliJ IDEA. En este caso se utilizó Apache Netbeans en su versión 15 junto con un JDK versión 11.0.17 para compilar el programa. No se utilizaron bibliotecas externar de Java y el programa fue ejecutado por cmd con sistema operativo Windows 10.

### **EJEMPLOS DE USO**

### **RESULTADOS ESPERADOS**

### **POSIBLES ERRORES**

## **RESULTADOS Y AUTOEVALUACIÓN**

### **RESULTADOS**

### **2.4.2 AUTOEVALUACIÓN**

La Autoevaluación se realiza de la siguiente forma: 0: No realizado – 0.25: Funciona 25% de las veces – 0.5: Funciona 50% de las veces 0.75: Funciona 75% de las veces – 1: Funciona 100% de las veces. Para ver la tabla de Autoevaluación, XXX

# **CONCLUSIÓN**

# **BIBLIOGRAFÍA Y REFERENCIAS**

# **ANEXOS**