Universidad de Santiago de Chile

Facultad de Ingeniería Informática

Paradigmas de Programación

Profesor Roberto González

Laboratorio 3
Programación Orientada a Objetos

Nicolás Aguilera

Diciembre 2022

Índice

1. Introducción

El siguiente informe corresponde al desarrollo del Laboratorio 3 del curso *Paradigmas de Programación*, el cual consiste en la aplicación de la *Programación Orientada a Objetos* (POO) a la resolución de un problema específico mediante el uso del lenguaje *Java*.

1.1. Descripción del problema

El problema consiste en la implementación de clases para el tratamiento imágenes RGB-D, similar a herramientas como GIMP y Adobe Photoshop, pero de forma simplificada. Para esto, se considera que las imagenes tienen un alto y un ancho (width x height) y que están compuestas por tres tipos de pixeles diferentes, pudiendo formar tres tipos de imagenes diferentes:

- **bitmap-d:** El valor que pueden tomar sus pixeles es de 0 o 1, indicando que el pixel se encuentra apagado o encendido (negro o blanco).
- **pixmap-d:** Los pixeles tienen tres canales de colores (R, G y B) los cuales toman valores de 0 a 255.
- **hexmap-d:** Los pixeles expresan sus canales RGB en formato hexadecimal.

La posición de cada pixel dentro de la imagen y la profundidad corresponden a valores enteros entre cero y el máximo, dependiendo de las dimensiones de la imagen y la profundidad de esta.

1.2. Descripción del paradigma

POO se enfoca en la creacíon clases, las cuales son una abstracción tal y como dice el nombre del paradgima, de objetos del mundo real, cada una de las cuales tiene atributos (características que definen al objeto) y métodos (funciones de estos objetos) que pueden ser instanciadas para crear objetos. Estos pueden interactuar entre ellos a través de estos métodos teniendo un grado de relación definido en el Diagrama de Clases. A diferencia de los paradigmas estudiados anteriormente, los TDAs implementados a través de este, cuentan con un simil a las funciones y relaciones (de los paradigmas funcional y lógico), que son los métodos mencionados anteriormente. A continuación se presentan algunos de los conceptos más importantes del paradigma:

- **Clases:** Son el molde de los objetos que se instancian a partir de estas y son definidas por sus atributos y métodos, pudiendo ser *abstractas*.
- **Objetos:** Son instancias de una clase.
- **Herencia:** Hay ciertas clases que pueden heredar atributos y métodos de otras si comparten gran parte de sus características o son una variación de la clase padre.

- **Polimorfismo:** A través de la herencia una clase se puede tener variaciones que contenga los mismos atributos y métodos, como por ejemplo: una clase Figura y clases Cuadrado, Ciruclo, Triángulo que hereden los atributos y métodos de Figura.
- **Encapsulamiento:** Los atributos y métodos de las clases pueden ser visibles o no para otras clases, esto a través del encapsulamiento, el cual puede ser *público*, *protegido* o *privado*.
- **Interface:** Es la representación del TDA y por lo tanto un molde para la creación de clases, las cuales implementan los métodos de la Interface.

2. Desarrollo

2.1. Análisis del problema

El foco principal del problema es el tratamiento de imágenes a través de la creación de clases que modelen los elementos involucrados en el problema, con sus atributos y métodos, y la interacción entre estas. A continuación se mencionan los principales elementos que conforman el problema:

- **Imagen:** Unidad fundamental del problema. Está constituida por un alto y ancho, así como por pixeles, los cuales deben responder a las dimensiones de la imagen (alto x ancho) y pueden ser de 3 tipos: Bitmap, Pixmap o Hexmap.
- **Pixel:** Las imágenes están conformadas por sub-unidades llamadas *pixeles*, los cuales pueden ser de diferentes tipos: pixbit-d, pixrgb-d y pixhex-d, conformando los diferentes tipos de imagen. Los pixeles a su vez, tienen una posición dentro de la imagen representada por coordendas x e y, un color (especificado anteriormente en *Descripción del problema*) y una profundidad.
- **Color:** Cada pixel tiene un color el cual es representado por un valor entero (o varios en el caso del pixrgb-d) o una cadena de caracteres (en hexadecimal) (en el caso de los pixhex-d).
- **Profundidad:** Además, los pixeles tienen una profundidad, que representa la "lejanía.[a] la que se encuentra ese pixel, desde el plano de la pantalla u hoja. Esta característica ayuda a la representación tridimensional de una imagen.

2.2. Diseño de la solución

2.2.1. TDAs implementados

Tal y como se mencionó anteriormente, la solución para el tratamiento de imágenes recae en una buena abstracción y representación los TDAs que son parte del problema, de tal manera que la implementación de la solución sea lo más sencilla posible y no caiga en redundancias. Esto significa que cada TDA debe estar compuesto por *Constructores*,

Getters, *Setters* y *Modificadores* cumpliendo cada uno sus roles específicos a la hora de implementar un TDA. A continuación se describen los TDAs considerados[1] para la solución del problema y parte de su implementación:

- **TDA Image:** Su representación está dada por los siguientes atributos: el **ancho** (width), el **alto** (height), el **color comprimido** de la imagen, una **lista de pixeles** que componen la imagen y por último una lista de **pixeles comprimidos** que contiene información reducida de los pixeles que no están en la imagen y será utilizada para su posterior recuperación (descompresión).
- **TDA Pixel:** Su representación está dada por una lista que contiene los siguientes elementos: una lista de dos elementos (**posición** x e y del pixel en la imagen), **color** del pixel (representación binaria, canales RGB y representación RGB hexadecimal) y **profundidad** del pixel.
- **TDA Histogram:** Su representación está dada por una lista de enteros que representa **frecuencias** y una lista de colores de la imagen asociado a la lista de frecuencias.

Debido a que todos los tipos de pixeles contienen los mismos parámetros (posición, color y profundidad), se decidió considerar el **TDA Pixel** como superclase para la representación de los distintos tipos de pixeles. Esto con el fin de acceder de manera más sencilla a cada uno de los argumentos del pixel. Para un detalle más exhaustivo del diseño de la solución, ver la Figura 1 del Anexo, la cual muestra el **Diagrama de Clases**, donde se observa la solución propuesta y la relación entre las clases implementadas.

2.2.2. Algoritmos y técnicas empleadas

Tal y como se especificó en el material entregado para el desarrollo del proyecto, todos los TDAs y algoritmos asociados se basaron en la creación de clases y su interacción a través de métodos, por lo que los subproblemas que requerían de modificaciones de las clases, utilizaron en su mayoría métodos iterativos, recorriendo arreglos y utilizando los métodos de cada clase.

2.3. Aspectos de implementación

2.3.1. Estructura del proyecto

Se utilizaron archivos diferentes para los TDAs y clases implementadas[2], en este caso una interface ImageInterface para la superclase Image y clases para sus diferentes tipos. Lo mismo con la superclase Pixel y sus diferentes tipos. Además se creó la superclase Histogram y clases para sus diferentes tipos. Por último, se creó la clase Menu, encargada

[1] Dado que las instrucciones del laboratorio solo especificaban realizar el TDA para Image, solo se realizó la Interface para este TDA, siendo el resto de TDA implementados directamente a través de clases.

[2] Todas las Interfaces y Clases implementadas tienen extensión **.java** y además del nombre tienen el formato Clase_RUT_Apellidos.java

de realizar las tareas de la interfaz por consola y la clase App, encargada de ejecutar todo lo anterior.

2.3.2. Compilador y bibliotecas utilizadas

El lenguaje de programación utilizado es *Java* en su versión 11.0.17, del cual se utilizaron los paquetes nativos disponibles, como parte de las exigencias del Laboratorio, en particular, el paquete *util*. Por otra parte el compilador que se usó fue *Gradle*.

2.4. Instrucciones de uso

Para hacer uso de la solución propuesta, se deben tener los archivos de Interface y Clase ante mencionados junto con las carpetas asociadas a *Gradle* para poder ejecutarlo, la Figura 5 del Anexo se muestra un ejemplo de carpeta. Luego, abrir la terminal en el directorio principal del proyecto y ejecutar lo siguiente:

- gradle run (GNU/Linux)
- gradlew.bat run (Windows)

Luego, interactuar con la terminal a través del teclado, tal y como indica el programa. Para facilitar el uso del programa, este viene con una imagen precargada. Una vez ejecutado el comando, se puede comenzar a realizar entradas por teclado y aplicar métodos a la imagen. ¿Cómo saber con qué imagen se está trabajando? El programa indica el tipo y dimensiones de la imagen con la que se está trabajando actualmente, pudiendo trabajar solamente con una imagen a la vez. Algunos ejemplos de la interacción con la terminal pueden encontrarse en las Figuras 2, 3 y 4 de la Sección 2 del *Anexo*.

2.4.1. Resultados esperados

Se espera que el usuario pueda crear, modificar y visualizar imágenes mediante la interfaz por consola, logrando rotar, voltear, comprimir e incluso mostrar los pixeles en pantalla, entre otras, visualizando los resultados en la terminal o en *Netbeans* si se compila y corre en esa IDE.

2.4.2. Posibles errores

Dado que se asumió que el usuario introducirá datos de forma adecuada y por lo tanto no introducirá valores fuera de los límites o tipos de datos inadecuados, puede que al hacer esto, las imágenes que cree el usuario y las modificaciones que le haga, provoquen errores que se mostrarán en la consola.

2.5. Resultados y autoevaluación

En general, los resultados obtenidos son los esperados, logrando la creación y manipulación de imágenes a través de la interacción con la interfaz por consola.

Los criterios y resultados de la *Autoevaluación* de requisitos funcionales pueden encontrarse en los Cuadros 5 y 6 del Anexo.

2.5.1. Métodos no completados

El único método que no se completó fue decompress de la clase Image.

3. Conclusión

Luego del desarrollo del Laboratorio, se puede concluir que se cumplió con los requisitos generales y particulares del proyecto. En primer lugar, construir un diagrama de clases para la abstraccción e implementación de los TDAs solicitados, logrando cohesión para los atributos y métodos de cada clase, otorgando la responsabilidad pertinente a cada una de estas, todo esto utilizando la programación orientada a objetos.

En cuanto a las dificultades para utilizar el paradigma lógico, a diferencia de los Laboratorios 1 y 2, se encuentran los pasos previos a la implementación, es decir la etapa de análisis y diseño, en cuanto a la creación del diagrama de clases. Por otro lado se encuentra la adaptación con el IDE *Netbeans* y la compilación con Gradle que me ocasionó muchos problemas con algunos errores en el build.

A diferencia de los anteriores laboratorios, la solución propuesta para este laboratorio fue diferente en cuanto a los métodos, ya que el lenguaje otorga otros recursos que en la programación funcional y lógica no se pueden utilizar, como los procesos iterativos, entre otros.

4. Bibliografía y referencias

1. Gacitúa D. (2022). *Guía para Proyecto Java usando Gradle*. Recuperado de: Guía Proyecto Java usando Gradle

2. González R. (2022). *Paradigmas de Programación: Proyecto semestral de laboratorio*. Recuperado de: Enunciado General del Proyecto Semestral

3. González R. (2022). *Paradigmas de Programación: Proyecto semestral de laboratorio. Laboratorio 3*. Recuperado de: Enunciado Laboratorio 3

5. Anexo

5.1. Sección 1

TDA Image - Requerimientos		
Nombre	**Tipo de método**	**Descripción**
Image	Constructor	Crea una instancia de la clase Image
isBitmap	Pertenencia	Determina si una imagen es tipo bitmap-d.
isPixmap	Pertenencia	Determina si una imagen es tipo pixmap-d.
isHexmap	Pertenencia	Determina si una imagen es tipo hexmap-d.
isCompressed	Pertenencia	Determina si una imagen está comprimida.
flipH	Modificador	Voltea una imagen de forma horizontal
flipV	Modificador	Voltea una imagen de forma vertical
crop	Modificador	Recorta una imagen dado un cuadrante determinado por dos puntos (x_1, y_1) y (x_2, y_2)
imgRGBtoHex	Modificador	Transforma una imagen tipo pixmap-d en una tipo hexmap-d
rotate90	Modificador	Rota una imagen 90 grados en sentido horario
compress	Modificador	Comprime una imagen eliminando los pixeles con el color más frecuente
changePixel	Modificador	Cambia un pixel de una imagen por otro.
invertColorRGB	Modificador	Cambia el valor de los canales RGB al valor simétricamente opuesto
invertColorBit	Modificador	Invierto los bits de cada pixel
imageToString	Otras	Muestra una imagen con sus pixeles representados como string
depthLayers	Otras	Crea una lista de imagenes que contienen pixeles de la misma profundidad
histogram	Otras	Entrega una lista de los colores que componen la imagen junto a su frecuencia

Cuadro 1: TDA Image y sus requerimientos funcionales.

TDA Image - Predicados adicionales		
Nombre	**Tipo de predicado**	**Descripción**
sortPixlist	Modificador	Ordena la lista de pixeles de la imagen.

Cuadro 2: Funciones adicionales del TDA Image.

Evaluación	Descripción
0	No se cumple o no funciona
0.25	Falla la mayoría de las veces (funciona el 25 % de las veces)
0.5	Funcionamiento irregular (50 % de las veces)
0.75	Funcionamiento con problemas menores (75 % de las veces funciona)
1	Se cumple o funciona todo el tiempo

Cuadro 3: Evaluación y descripción utilizada para la Autoevaluación.

Función	Evaluación
TDAs	1
Image	1
isBitmap	1
isPixmap	1
isHexmap	1
isCompressed	1
flipH	1
flipV	1
crop	1
imgRGBtoHex	1
histogram	1
rotate90	1
compress	1
changePixel	1
invertColorBit	1
invertColorRGB	1
imageToString	1
depthLayers	1
decompress	0

Cuadro 4: Requerimientos funcionales y su evaluación.

5.2. Sección 2

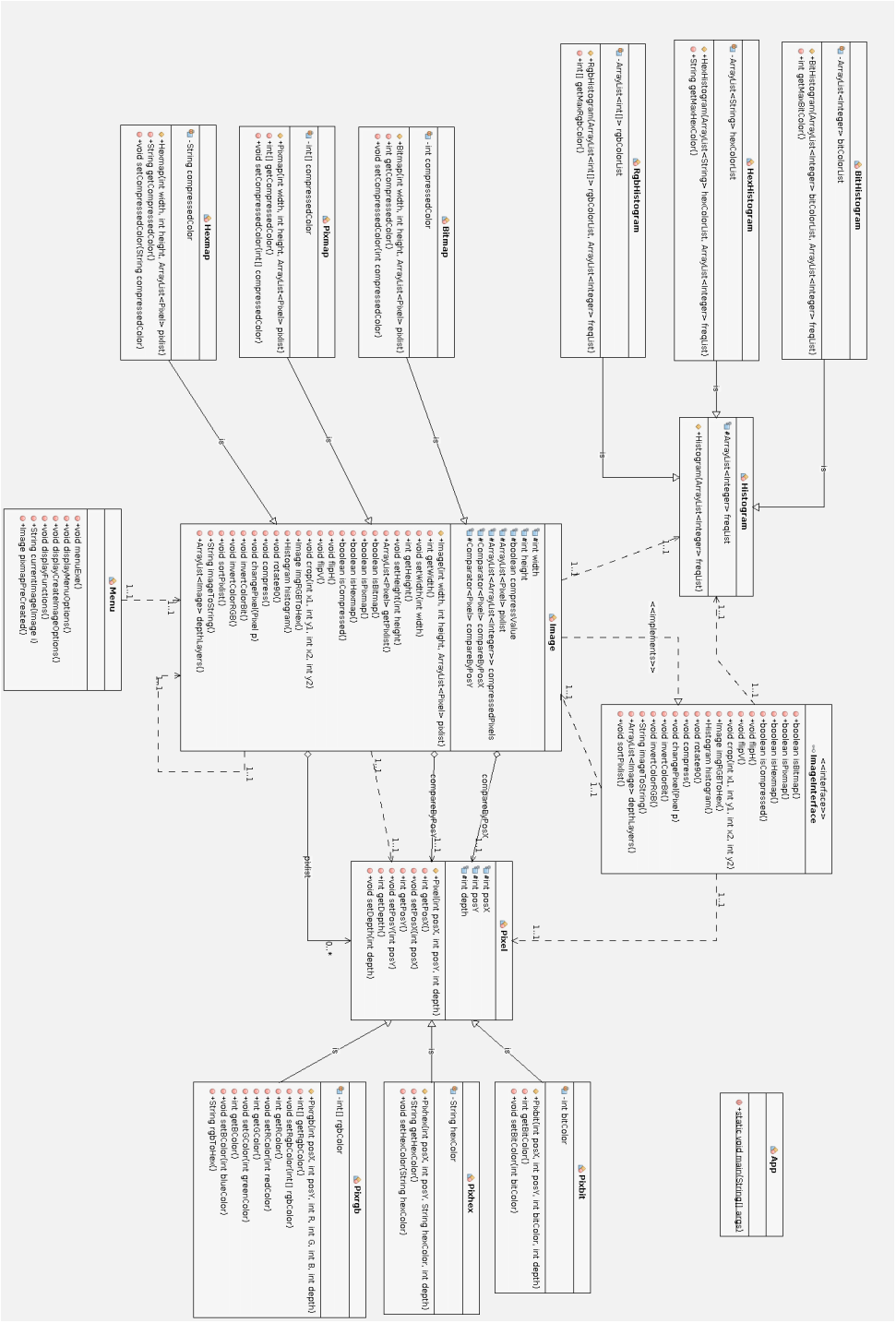

Figura 1: Diagrama de Clases del problema generado a través de NetBeans.

```
Image Actual: Pixmap de 3x2
1. Crear imagen
2. Modificar imagen
3. Visualizar imagen
4. Salir

Introduzca su opción: 2

Selecciona la modificación a realizar:
1. Voltear Horizontalmente
2. Voltear Verticalmente
3. Recortar
4. Cambiar de RGB a Hexadecimal
5. Voltear 90 sentido horario
6. Comprimir imagen
7. Cambiar un pixel
8. Invertir Bits
9. Invertir canales RGB
10. Capas de profundidad
11. Verificar Bitmap
12. Verificar Pixmap
13. Verificar Hexmap
14. Verificar Compresión

15. Volver a Menú Principal

Introduzca su opción: 2

Acción realizada exitosamente.
```

Figura 2: Ejemplo de interacción con la interfaz para modificar una imagen.

```
Image Actual: Pixmap de 3x2
1. Crear imagen
2. Modificar imagen
3. Visualizar imagen
4. Salir

Introduzca su opción: 3
[45,20,20] [20,200,120] [70,23,67]
[20,200,120] [20,200,120] [60,180,100]


Image Actual: Pixmap de 3x2
1. Crear imagen
2. Modificar imagen
3. Visualizar imagen
4. Salir

Introduzca su opción: 4
Has salido.
```

Figura 3: Ejemplo de interacción con la interfaz para visualizar la imagen.

```
Elija el tipo de imagen que desea crear:
1. Bitmap
2. Pixmap
3. Hexmap
Introduzca su opción: 1
Introduzca el ancho: 2
Introduzca el largo: 2

Pixel N°1
Ingrese posición X: 0
Ingrese posición Y: 0
Ingrese valor del Bit: 1
Ingrese la profundidad: 2
Pixel N°2
Ingrese posición X: 0
Ingrese posición Y: 1
Ingrese valor del Bit: 1
Ingrese la profundidad: 3
Pixel N°3
Ingrese posición X: 1
Ingrese posición Y: 0
Ingrese valor del Bit: 0
Ingrese la profundidad: 2
Pixel N°4
Ingrese posición X: 1
Ingrese posición Y: 1
Ingrese valor del Bit: 1
Ingrese la profundidad: 1
Imagen creada con éxito!
```

Figura 4: Ejemplo de creación de imagen tipo Bitmap.