Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

Proyecto de Laboratorio 2022 - Paradigma Orientado a Objetos

Paradigmas de Programación

Lucas Mesias Soza
Sección: 13310-0-A-1
Profesor: Roberto Gonzalez Ibañez

Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

Índice

Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

1. Introducción

Un programa de edición de imágenes permite modificar o alterar imágenes existentes y crear los elementos visuales con los que interactuamos día a día, como una simple fotografía con filtros simples o ajustes de color, afiches, eliminación de fondos, o directamente el programa se usa como herramienta de dibujo digital, programas como Adobe Photoshop, GIMP, Photopea, entre otros, son las herramientas con las que se trabaja para crear este mundo digital al que estamos acostumbrados.

2. Problema

Se requiere crear un software de manipulación de imágenes simplificado en el lenguaje Java, bajo el paradigma orientado a objetos, creando las distintas clases que conformarán el programa, usando una estructura de clases adecuada al problema, donde principalmente se deben trabajar los 3 tipos de pixeles y 3 tipos de imágenes desde clases distintas, con las que se pueda trabajar genéricamente como imagen. Adicionalmente se incluye como requerimiento opcional una interfaz gráfica, la cual no se implementó.

Los pixeles serán la base del trabajo de manipulación, estos deben ser representados en un TDA que permita guardar la siguiente información: posición, valor de color y profundidad. Las imágenes se deben representar en un TDA adecuado, que permita trabajar pixel a pixel, además habrán 3 tipos distintos de pixeles, diferenciandose en el valor de color que almacenan, estos serán bit (0 o 1), RGB y hexadecimal, además entre las operaciones que debe tener el software, debe ser capaz de comprimir y descomprimir imágenes, por lo que la implementación de la imágen debe soportar estas características.

3. Solución

Los TDAs implementados para los pixeles fueron estructurados en un TDA principal llamado “pixel”, el cual almacena el tipo de pixel, las coordenadas X, Y, el valor de color, y la profundidad en una lista. Luego, se crearon 3 sub-TDAs, uno por cada tipo de valor de color, “pixbit” para almacenar el número que representa el bit, “pixrgb” para almacenar 3 números enteros con los valores (R G B) y “pixhex” para almacenar el valor RGB en formato hexadecimal “#RRGGBB”, cada uno de estos es un pixel que guarda información diferente, de esta forma, se pueden implementar predicados básicos para pixeles genéricos, simplificando la implementación de predicados que no dependen del tipo de pixel. El TDA imagen guarda las dimensiones de la imágen, un valor de compresión, una lista de pixeles (los cuales no necesitan estar en un orden particular) y la imagen misma, los métodos implementados no dependen del orden de esta lista, pero todos los pixeles de la imagen deben estar presentes o habrá inconsistencias a la hora de la descompresión.

Se crearon diagramas UML antes y después de la realización del laboratorio, ambos presentes en el anexo. Estos representan la estructura de las clases que conforman el código, al terminar el laboratorio el diagrama inicial cambió considerablemente.

Para los métodos que identifican el tipo de imagen se revisa que cada pixel de la imagen sea del tipo correcto, ya que el usuario debe introducir estos de manera homogénea.

Histograma: Para hexmap y pixmap se extraen todos los valores de color presentes en la imagen y se almacenan en una lista, la cual se ordena sin eliminar duplicados, luego, se ordena la lista y se cuentan las repeticiones del color, almacenando temporalmente 2 cantidades a la vez, comparando ambas y guardando el color con más repeticiones, finalmente, se guarda el valor de compresión y la cantidad de veces que se repite en una string de formato "COLOR/CANTIDAD".

Compresión: Usando la información entregada por Histograma, se elimina de la imagen los pixeles con el color más común recorriendo iterativamente la lista de pixeles de la imagen, luego se guarda en el valor de compresión el color eliminado.

Descompresión: Al ingresar una imagen comprimida, se rellenan los pixeles que faltan en la imagen con pixeles nuevos, con el valor de compresión haciendo uso de un método recursivo *findPix*. Cabe destacar que con esta implementación se pierde totalmente la información de profundidad en los pixeles comprimidos.

Transformación de imagen a string: Se ordena la lista de pixeles, primero por la coordenada X de los pixeles y luego por la coordenada Y, una vez ordenados los pixeles se transforma el color de cada uno a string y se concatena adecuadamente.

Capas de Profundidad: Primero se extrae de la imagen todas las profundidades de los pixeles sin repetición y se guardan, luego se genera una imagen por cada profundidad, y se recorren los pixeles, cuando se encuentra un pixel de la profundidad deseada, se almacena sin modificaciones, pero cuando ocurre el caso contrario, se cambia por un pixel blanco. Para simplificar esta función, se implementó una función por sobreescritura en la clase pixel, luego en cada tipo de pixel se implementa un método que genera un pixel de las mismas coordenadas y profundidad, pero de color blanco del tipo adecuado.

El proyecto se organizó en 9 clases, 2 clases padre, 6 heredadas, y 1 interfaz. No se usó ninguna librería externa, y se usó gradle para compilar el proyecto.

4. Instrucciones de uso

Se requiere java jdk 11.0.17 instalado. Se usará la herramienta Gradle para compilar y ejecutar el laboratorio.
Desde una terminal en la carpeta descargada del proyecto se ejecuta el comando "*./gradle.bat run –console=plain*", el proyecto comenzará a ejecutarse y el usuario podrá interactuar con el menú escribiendo la opción deseada y presionando ***enter*** cuando se indique, el programa usará una única imagen que será sobreescrita en cada modificación realizada. Se adjuntan ejemplos de uso en el anexo

5. Resultados

Se lograron implementar todas las funciones tanto obligatorias como opcionales, exceptuando la interfaz gráfica, todas funcionan si la entrada es correcta, aunque se pudo haber hecho un esfuerzo mayor y revisar que las entradas sean correctas antes de procesar, este aspecto no fue una prioridad en el desarrollo del proyecto. Adicionalmente es posible la optimización del

proyecto en varios aspectos, memoria, cambiando el identificador de tipo de pixel de string a un entero y representar los tipos en base a una enumeración, cambiar el uso de *append* por *cons*, entre otras mejoras.

6. Autoevaluación

La autoevaluación del proyecto con respecto a las funciones implementadas es de puntaje completo, la implementación del menú revisa si la operación se puede realizar en el tipo de imagen actual, si no es posible se envía una advertencia al usuario y no se ejecuta la acción, exceptuando la visualización de una imagen comprimida, en este caso se le indica al usuario que la imagen está comprimida, por lo tanto, habrá fallos al intentar visualizarla sin descomprimir antes.

7. Conclusiones

El trabajo realizado en java fue simple y sencillo de implementar, es muy similar a C o Python, por lo que desarrollar los métodos no implicó mayor inconveniencia, pero la estructuración del proyecto fue más difícil de lo esperado, ya que la estructura general cambió varias veces en el desarrollo de este proyecto, a pesar de que los UML inicial y final son muy similares.

8. Anexo

Diagrama UML inicial o de análisis

Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

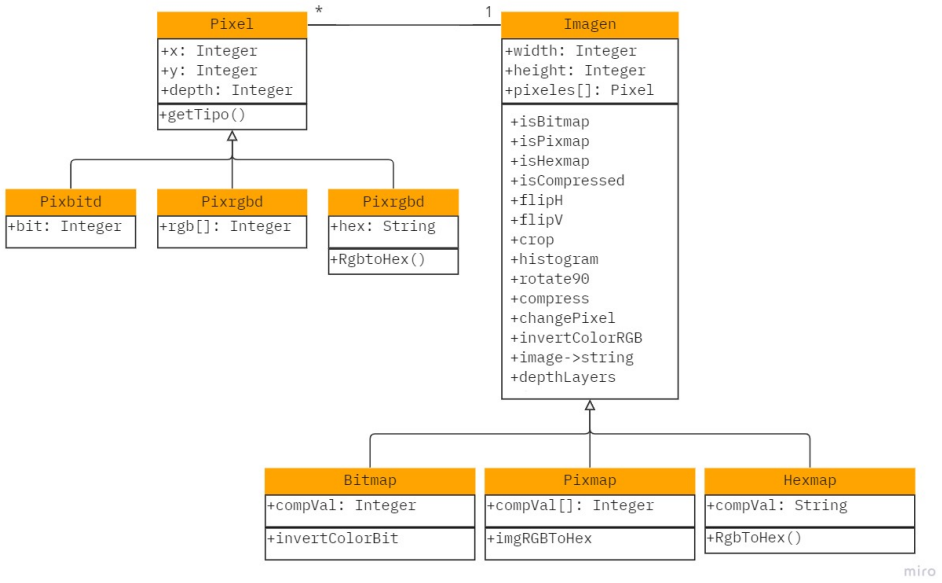
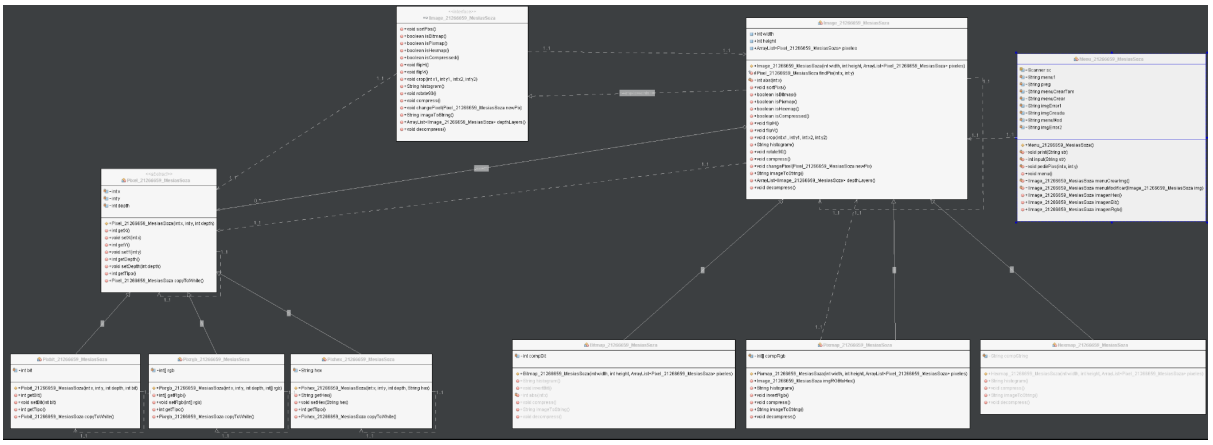
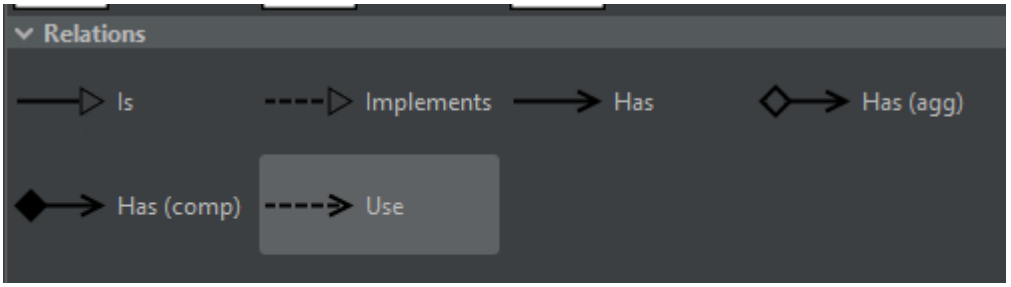


Diagrama UML final o de diseño



Leyenda UML de diseño



Puntajes Autoevaluación:

Requerimientos funcionales	Nota
Clases y TDAs	1
Menu	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
ToString	1
DepthLayers	1
Decompress	1

Ejemplos de uso:

```
Opciones:
1.Crear Imagen
2.Modificar Imagen
3.Visualizar Imagen
4.Imagen Bitmap Predeterminada
5.Imagen Pixmap Predeterminada
6.Imagen Hexmap Predeterminada

Introduzca su opci├│n:4

Opciones:
1.Crear Imagen
2.Modificar Imagen
3.Visualizar Imagen
4.Imagen Bitmap Predeterminada
5.Imagen Pixmap Predeterminada
6.Imagen Hexmap Predeterminada

Introduzca su opci├│n:3
0 1 0
1 0 1
0 1 0
```

```
Opciones:
1.Crear Imagen
2.Modificar Imagen
3.Visualizar Imagen
4.Imagen Bitmap Predeterminada
5.Imagen Pixmap Predeterminada
6.Imagen Hexmap Predeterminada

Introduzca su opci├│n:2
1.Flip Horizontal
2.Flip Vertical
3.Recortar
4.Rotar 90 grados
5.Histograma (Color que m├ís se reptite)
6.Comprimir
7.Cambiar Pixel
8.Capas de Profundidad
9.Descomprimir
10.Invertir Bits
11.Invertir RGB
12.Comprobar Bitmap
13.Comprobar Pixmap
14.Comprobar Hexmap
13.Transformar Pixmap a Hexmap

Introduzca su opci├│n:10

Opciones:
1.Crear Imagen
2.Modificar Imagen
3.Visualizar Imagen
4.Imagen Bitmap Predeterminada
5.Imagen Pixmap Predeterminada
6.Imagen Hexmap Predeterminada

Introduzca su opci├│n:3
1 0 1
0 1 0
1 0 1
```