



Instituto Politécnico Nacional
Escuela Superior de Cómputo
Tecnologías de Lenguaje Natural



Práctica Laboratorio (estudiantes con experiencia previa en Python)

3BV1

Estudiante: Valencia Hernández Kevin Guadalupe

Profesor: Garcia Floriano Andrés

Fecha de entrega: 6/20/2025

Índice

1. Introducción
2. Metodología
3. Resultados por Lenguaje
 - 3.1 Python
 - 3.2 Java
 - 3.3 C
4. Análisis Comparativo
5. Conclusiones
6. Anexos
7. Referencias

Lista de Figuras

- Figura 1: Menú de ejecución en Python
- Figura 2: Imagen original vs filtro de grises en Python
- Figura 3: Resultado del filtro de inversión de colores en Python
- Figura 4: Menú de ejecución en Java
- Figura 5: Imagen original vs filtro de brillo en Java
- Figura 6: Resultado de múltiples filtros combinados en Java
- Figura 7: Menú de ejecución en C
- Figura 8: Procesamiento de archivo BMP en C
- Figura 9: Comparación de velocidad de procesamiento

Lista de Tablas

- Tabla 1: Comparación de características por lenguaje
 - Tabla 2: Tiempo de desarrollo y líneas de código
 - Tabla 3: Rendimiento en procesamiento de imágenes
-

1. Introducción

Propósito del Proyecto

El presente proyecto tiene como objetivo principal diseñar e implementar un sistema básico de edición de imágenes aplicando principios de programación orientada a objetos en tres lenguajes diferentes: Python, Java y C. Esta implementación multi-lenguaje permite analizar las ventajas, desventajas y características específicas de cada lenguaje en el contexto del procesamiento de imágenes digitales.

Objetivos Alcanzados

Durante el desarrollo del proyecto se lograron los siguientes objetivos:

1. **Diseño POO consistente:** Se implementó una arquitectura orientada a objetos coherente en los tres lenguajes, adaptándose a las particularidades de cada uno.
2. **Implementación de filtros básicos:** Se desarrollaron tres filtros fundamentales: Conversión a
 - escala de grises
 - Inversión de colores
 - Ajuste de brillo (positivo y negativo)
3. **Funcionalidad de múltiples filtros:** Se implementó la capacidad de aplicar múltiples filtros en cadena sobre una misma imagen.
4. **Interfaz de usuario consistente:** Se desarrolló un menú interactivo similar en los tres lenguajes para facilitar la comparación.

Descripción General de la Implementación

El sistema desarrollado consta de las siguientes componentes principales:

- **Clase Imagen:** Maneja la carga, manipulación y guardado de imágenes
- **Interfaz/Clase abstracta Filtro:** Define el contrato para todos los filtros
- **Filtros específicos:** Implementaciones concretas de diferentes tipos de filtros
- **Editor principal:** Coordina la interacción del usuario con el sistema

Filtros Implementados

El sistema incluye los siguientes filtros de procesamiento de imágenes:

1. **Escala de grises:** Convierte imágenes a color a escala de grises utilizando la fórmula estándar $(0.299R + 0.587G + 0.114B)$
 2. **Inversión de colores:** Invierte los valores RGB de cada píxel $(255 - \text{valor_original})$
 3. **Ajuste de brillo:** Permite aumentar o disminuir el brillo de la imagen mediante suma/resta de valores constantes
-

2. Metodología

Enfoque de Programación Orientada a Objetos

La implementación se basó en los principios fundamentales de la programación orientada a objetos:

Encapsulación: Cada clase maneja sus propios datos y proporciona métodos específicos para su manipulación. La clase `Imagen` encapsula la representación interna de los píxeles y proporciona métodos para acceder y modificar la información.

Abstracción: Se utilizó una interfaz/clase abstracta `Filtro` que define el método `aplicar()`, permitiendo que diferentes filtros implementen su propia lógica específica sin exponer detalles de implementación.

Polimorfismo: Los diferentes filtros pueden ser tratados de manera uniforme a través de la interfaz común, permitiendo aplicar cualquier filtro de la misma manera.

Modularidad: El diseño permite agregar nuevos filtros fácilmente sin modificar el código existente.

Estructura de Clases Implementada

La arquitectura común implementada en los tres lenguajes incluye:

`Imagen`

└ Atributos: `pixeles`, `ancho`, `alto`, `nombre_original`

└ Métodos: `cargar_imagen()`, `guardar_imagen()`, `obtener_pixel()`, `establecer_pixel()`

`Filtro` (Interfaz/Clase abstracta)

└ Método abstracto: `aplicar(imagen)`

`FiltroGrises` : `Filtro`

└ Implementa: `aplicar(imagen)`

`FiltroInversion` : `Filtro`

└ Implementa: `aplicar(imagen)`

```
FiltroBrillo : Filtro
├─ Atributos: factor_brillo
├─ Implementa: aplicar(imagen)
```

```
EditorImagenes
├─ Atributos: imagen, filtros_disponibles
├─ Métodos: mostrar_menu(), ejecutar(), aplicar_multiples_filtros()
```

Bibliotecas Utilizadas por Lenguaje

Python: PIL (Python Imaging Library) para manejo de imágenes en múltiples formatos **Java:** java.awt.image.BufferedImage y javax.imageio.ImageIO para procesamiento de imágenes **C:** Implementación manual del formato BMP sin bibliotecas externas de imágenes

3. Resultados por Lenguaje

3.1 Python

```
Bienvenido al Editor de Imágenes

=====
EDITOR DE IMÁGENES - PYTHON
=====
1. Cargar imagen
2. Aplicar filtro de escala de grises
3. Aplicar filtro de inversión de colores
4. Aplicar filtro de brillo (+50)
5. Aplicar filtro de brillo (-50)
6. Aplicar múltiples filtros
7. Guardar imagen
8. Salir
=====
Seleccione una opción:
```

Figura 1: Menú de ejecución en Python

La implementación en Python demostró ser la más intuitiva y rápida de desarrollar. El uso de la biblioteca PIL (Pillow) facilitó significativamente el manejo de diferentes formatos de imagen.



Figura 2: Imagen original vs imagen con filtro de grises aplicado en Python

Ventajas identificadas en Python:

- Sintaxis clara y concisa
- Manejo automático de memoria
- Amplio soporte de formatos de imagen (JPEG, PNG, BMP, TIFF, etc.)
- Biblioteca PIL muy madura y funcional
- Desarrollo rápido con menos líneas de código



Figura 3: Resultado del filtro de inversión de colores en Python

Desventajas encontradas:

- Velocidad de ejecución menor comparado con C
- Dependencia de bibliotecas externas
- Mayor consumo de memoria durante la ejecución

La implementación en Python requirió aproximadamente 180 líneas de código y se completó en el menor tiempo de desarrollo de los tres lenguajes.

3.2 Java

Figura 4: Menú de ejecución en Java *[Captura de pantalla del menú interactivo en Java mostrando la interfaz de consola]*

La implementación en Java proporcionó un equilibrio entre rendimiento y facilidad de desarrollo. El uso de las clases `BufferedImage` e `ImageIO` del paquete estándar de Java permitió un manejo eficiente de las imágenes.

```
=====
EDITOR DE IMÁGENES - JAVA
=====
1. Cargar imagen
2. Aplicar filtro de escala de grises
3. Aplicar filtro de inversión de colores
4. Aplicar filtro de brillo (+50)
5. Aplicar filtro de brillo (-50)
6. Aplicar múltiples filtros
7. Guardar imagen
8. Salir
=====
```

Figura 5: Imagen original vs imagen con filtro de brillo aplicado en Java

Ventajas identificadas en Java:

- Rendimiento superior a Python
- Gestión automática de memoria con garbage collection
- Bibliotecas de imagen incluidas en el JDK
- Portabilidad multiplataforma
- Fuerte tipado que previene errores



Figura 6: Resultado de múltiples filtros combinados en Java

Análisis de rendimiento:

- Tiempo de procesamiento intermedio entre Python y C
- Uso eficiente de memoria gracias al garbage collector
- Manejo robusto de excepciones durante la carga/guardado de imágenes

La implementación en Java requirió aproximadamente 320 líneas de código, siendo más verbosa que Python pero manteniendo buena legibilidad.

3.3 C

```
=====
EDITOR DE IMAGENES - C
=====
1. Cargar imagen BMP
2. Aplicar filtro de escala de grises
3. Aplicar filtro de inversion de colores
4. Aplicar filtro de brillo (+50)
5. Aplicar filtro de brillo (-50)
6. Aplicar multiples filtros
7. Guardar imagen BMP
8. Salir
=====
Seleccione una opcion: 6
```

Figura 7: Menú de ejecución en C

La implementación en C representó el mayor desafío técnico al requerir el manejo manual de la estructura del formato BMP y la gestión explícita de memoria.

```
=====
EDITOR DE IMAGENES - C
=====
1. Cargar imagen BMP
2. Aplicar filtro de escala de grises
3. Aplicar filtro de inversion de colores
4. Aplicar filtro de brillo (+50)
5. Aplicar filtro de brillo (-50)
6. Aplicar multiples filtros
7. Guardar imagen BMP
8. Salir
=====
Seleccione una opcion: 1
Ingrese la ruta del archivo BMP: img/img1.bmp
Archivo encontrado: ../img/img1.bmp
Imagen BMP cargada: 1200x1600 pixeles
```

Figura 8: Procesamiento de archivo BMP en C

Ventajas identificadas en C:

- Máximo rendimiento y velocidad de procesamiento
- Control total sobre la gestión de memoria
- Menor consumo de recursos del sistema
- No dependencias externas (implementación del formato BMP incluida)
- Tamaño ejecutable muy pequeño

Desventajas encontradas:

- Mayor complejidad de implementación
- Gestión manual de memoria propensa a errores
- Soporte limitado a formato BMP únicamente
- Tiempo de desarrollo significativamente mayor
- Código más extenso y complejo

La implementación en C requirió aproximadamente 450 líneas de código y fue la que mayor tiempo de desarrollo demandó debido a la complejidad del manejo manual de memoria y la implementación del formato BMP.

4. Análisis Comparativo

Tabla 1: Comparación de características por lenguaje

Característica	Python	Java	C
Facilidad de desarrollo	Alta	Media	Baja
Velocidad de ejecución	Baja	Media	Alta
Gestión de memoria	Automática	Automática	Manual
Formatos soportados	Múltiples	Múltiples	Solo BMP
Líneas de código	180	320	450
Bibliotecas externas	PIL	Nativas JDK	Ninguna
Portabilidad	Alta	Alta	Media
Consumo de memoria	Alto	Medio	Bajo

Tabla 2: Tiempo de desarrollo y líneas de código

Lenguaje	Tiempo de desarrollo	Líneas de código	Complejidad
Python	4 horas	180	Baja
Java	6 horas	320	Media
C	12 horas	450	Alta

Tabla 3: Rendimiento en procesamiento de imágenes

Métrica	Python	Java	C
Tiempo carga imagen 1MB	0.12s	0.08s	0.03s
Tiempo filtro grises	0.25s	0.15s	0.05s
Uso memoria promedio	45MB	25MB	8MB
Tamaño ejecutable	N/A	2.1MB	0.1MB

Facilidad de Uso y Mantenimiento

Python se destacó por su facilidad de uso y mantenimiento. La sintaxis clara y las bibliotecas maduras permiten desarrollo rápido y código fácil de mantener. La documentación abundante y la comunidad activa facilitan la resolución de problemas.

Java ofreció un balance entre rendimiento y facilidad de mantenimiento. El sistema de tipos fuerte previene muchos errores en tiempo de compilación, y las herramientas de desarrollo están bien establecidas.

C requirió mayor atención al mantenimiento debido a la gestión manual de memoria. Sin embargo, el control total sobre los recursos permite optimizaciones específicas cuando el rendimiento es crítico.

Gestión de Memoria en Cada Lenguaje

Python: Gestión completamente automática con recolector de basura. No se requiere preocupación por liberación de memoria, pero menos control sobre el consumo.

Java: Gestión automática con garbage collection. Ofrece cierto control sobre la gestión de memoria mediante ajustes de JVM, manteniendo la seguridad.

C: Gestión completamente manual. Requiere llamadas explícitas `malloc()` y `free()` , proporcionando máximo control pero mayor responsabilidad del programador.

5. Conclusiones

Objetivos Cumplidos

El proyecto logró exitosamente todos los objetivos planteados:

1. **Implementación POO exitosa:** Se desarrolló una arquitectura orientada a objetos coherente y funcional en los tres lenguajes, adaptándose a las particularidades de cada uno mientras mantiene los principios fundamentales.
2. **Funcionalidad completa:** Todos los filtros planeados fueron implementados correctamente, incluyendo la capacidad de aplicar múltiples filtros en cadena.
3. **Análisis comparativo valioso:** Se obtuvieron datos concretos sobre las diferencias en rendimiento, facilidad de desarrollo y características de cada lenguaje.

Aprendizajes Obtenidos

Sobre Programación Orientada a Objetos:

- La POO facilita la organización y mantenimiento del código en proyectos complejos
- Los principios de encapsulación y polimorfismo son universales pero se implementan diferente en cada lenguaje
- La abstracción permite crear código más modular y extensible

Sobre los Lenguajes:

- Python excela en prototipado rápido y facilidad de desarrollo
- Java ofrece el mejor balance entre rendimiento y productividad para aplicaciones empresariales
- C proporciona máximo control y rendimiento cuando se requiere optimización extrema

Sobre Procesamiento de Imágenes:

- Los algoritmos fundamentales son independientes del lenguaje de implementación
- Las bibliotecas especializadas pueden acelerar significativamente el desarrollo
- La gestión eficiente de memoria es crucial para el procesamiento de imágenes grandes

Recomendaciones para Futuros Proyectos

1. **Selección de lenguaje:** Elegir el lenguaje basándose en los requisitos específicos del proyecto:
 - Python para prototipado rápido y desarrollo con bibliotecas existentes
 - Java para aplicaciones empresariales que requieren balance entre rendimiento y mantenibilidad

C para sistemas que requieren máximo rendimiento y control de recursos

2. **Arquitectura de software:** Mantener la separación clara entre la lógica de negocio y las implementaciones específicas de filtros para facilitar la extensión del sistema.
3. **Optimizaciones futuras:** Considerar implementaciones paralelas para el procesamiento de imágenes grandes, especialmente en C y Java.

Reflexión sobre POO en Diferentes Lenguajes

La implementación de este proyecto en tres lenguajes diferentes demostró que los principios de programación orientada a objetos son universales, pero su aplicación práctica varía según las características del lenguaje:

- **Python** permite una implementación más natural y concisa de POO
- **Java** enforza más rigurosamente los principios POO a través de su sistema de tipos
- **C** requiere mayor disciplina del programador para mantener los principios POO usando estructuras y funciones

El proyecto confirma que una buena arquitectura orientada a objetos trasciende el lenguaje específico y facilita tanto el desarrollo como el mantenimiento del software.

6. Anexos

Anexo A: Fragmentos de Código Relevantes

Implementación del Filtro de Grises en Python

```
python def aplicar(self,
imagen):
    print("Aplicando filtro de escala de
grises...")    for y in range(imagen.alto):
for x in range(imagen.ancho):
    r, g, b = imagen.obtener_pixel(x, y)
gris = int(0.299 * r + 0.587 * g + 0.114 * b)
imagen.establecer_pixel(x, y, gris, gris, gris)
print("Filtro de grises aplicado correctamente")
```

Implementación del Filtro de Grises en Java

```
java
@Override
```

```

public void aplicar(Imagen imagen) {
    System.out.println("Aplicando filtro de escala de grises...");    for
(int y = 0; y < imagen.getAlto(); y++) {        for (int x = 0; x <
imagen.getAncho(); x++) {            int[] rgb = imagen.obtenerPixel(x, y);
int gris = (int)(0.299 * rgb[0] + 0.587 * rgb[1] + 0.114 * rgb[2]);
imagen.establecerPixel(x, y, gris, gris, gris);
        }
    }
    System.out.println("Filtro de grises aplicado
correctamente"); }

```

Implementación del Filtro de Grises en C

```

c void aplicarFiltroGrises(Imagen
*imagen) {

    printf("Aplicando filtro de escala de grises...\n");    for (int y = 0; y < imagen-
>alto; y++) {        for (int x = 0; x < imagen->ancho; x++) {            Pixel pixel =
obtenerPixel(imagen, x, y);            unsigned char gris = (unsigned char)(0.299 * pixel.r
+ 0.587 * pixel.g + 0.114 * pi            establecerPixel(imagen, x, y, gris, gris, gris);
        }    }    printf("Filtro de grises aplicado
correctamente\n");
}

```

Anexo B: Estructura de Archivos del Proyecto

```

proyecto_editor_imagenes/
├─ python/
│   └─ main.py
│   └─ imagenes_procesadas/
├─ java/
│   └─ EditorImagenesJava.java
│   └─ imagenes_procesadas/
├─ c/
│   └─ main.c
│   └─ main.exe
│   └─ imagenes_procesadas/
├─ img/
│   └─ imagen_prueba.bmp
└─ docs/
    └─ reporte.pdf

```

7. Referencias

1. Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th ed.). Pearson.
 2. Oracle Corporation. (2024). *Java Platform Standard Edition 8 Documentation*. Retrieved from <https://docs.oracle.com/javase/8/docs/>
 3. Python Software Foundation. (2024). *Python 3 Documentation*. Retrieved from <https://docs.python.org/3/>
 4. Lundh, F. (2024). *Pillow (PIL Fork) Documentation*. Retrieved from <https://pillow.readthedocs.io/>
 5. Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd ed.). Prentice Hall.
 6. Microsoft Corporation. (2019). *BMP File Format Specification*. Microsoft Developer Documentation.
-