Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2025

ENSAYO SOBRE EL PARADIGMA PROCEDURAL VS EL PARADIGMA ORIENTADO A OBJETOS

NextGen Systems

202111134 Mario René Mérida Taracenaz
202200254 Karvin Armaldo Lago Pérez
202100171 Alex Oswaldo López Alquejay
202300684 Juan Amilcar Garcia Burrion
201903897 Luis Ivan Hidalgo Peña

Resumen

El paradigma procedural organiza el código en funciones que ejecutan instrucciones de forma secuencial, facilitando su comprensión en programas pequeños. Sus ventajas incluyen simplicidad y menor uso de memoria, pero su escalabilidad y organización pueden volverse problemáticas en proyectos grandes. Por otro lado, el paradigma orientado a objetos (OO) estructura el código en clases y objetos, promoviendo la reutilización, modularidad y mantenimiento eficiente. Sin embargo, introduce una mayor complejidad, requiere más memoria y genera mayor cantidad de código. La elección entre ambos paradigmas depende del tamaño y complejidad del proyecto, así como de la necesidad de escalabilidad y organización.

Palabras clave

Polimorfismo: Habilidad de diferentes clases para responder de manera diferente a un mismo método, mejorando la flexibilidad del código.
Funciones: Bloques de código reutilizables en el paradigma procedural que ejecutan tareas específicas.
Objetos: Instancias de una clase que encapsulan datos y comportamientos en el paradigma orientado a objetos.

Abstract

The procedural paradigm organizes code into functions that execute instructions sequentially, making it easier to understand in small programs. Its advantages include simplicity and lower memory usage, but its scalability and organization can become problematic in large projects. On the other hand, the object-oriented (OO) paradigm structures code into classes and objects, promoting reuse, modularity, and efficient maintenance. However, it introduces greater complexity, requires more memory, and generates a larger amount of code. The choice between the two paradigms depends on the size and complexity of the project, as well as the need for scalability and organization.

Keywords

Polymorphism: The ability of different classes to respond differently to the same method, improving code flexibility.
Functions: Reusable blocks of code in the procedural paradigm that perform specific tasks.
Objects: Instances of a class that encapsulate data and behavior in the object-oriented paradigm.

Introducción

En el desarrollo de software, la elección del paradigma de programación es un aspecto fundamental que influye en la eficiencia, escalabilidad y mantenimiento del código. Existen diversos enfoques, pero dos de los más utilizados son el paradigma procedural y el paradigma orientado a objetos (OO). Mientras que el primero se basa en la estructuración del código en funciones que ejecutan instrucciones de manera secuencial, el segundo organiza la lógica del programa en clases y objetos que encapsulan datos y comportamientos. Cada paradigma presenta ventajas y desventajas que los hacen más adecuados para ciertos tipos de proyectos. En este ensayo, se analizarán las principales características de ambos paradigmas con el objetivo de comprender su aplicabilidad y relevancia en el contexto del desarrollo de software moderno.

Desarrollo del Tema

Paradigma Procedural

El paradigma procedural es un enfoque de programación basado en la organización del código mediante funciones y procedimientos. Su estructura está centrada en la ejecución secuencial de instrucciones y la manipulación directa de datos.

Principios clave:

- Segmentación del código en funciones reutilizables.
- Enfoque en el flujo de ejecución de instrucciones.
- Uso de estructuras como variables, condicionales (if), y bucles (for, while).

Ventajas:

- Simplicidad y facilidad de implementación: Especialmente en programas pequeños, su comprensión y escritura resultan intuitivas.
- Menor uso de memoria: Al evitar la instanciación de objetos, se optimiza el uso de recursos.
- Código más corto y directo: Ideal para tareas específicas como la simplificación de fracciones.

Desventajas:

- Difícil de escalar: En programas grandes, el mantenimiento se vuelve complejo.
- Menor reutilización de código: Al no encapsular datos y comportamientos en objetos, se dificulta la reutilización.
- Estructura poco organizada: La proliferación de funciones aisladas puede derivar en una arquitectura desordenada.

Paradigma Orientado a Objetos (OO)

El paradigma orientado a objetos (OO) estructura el código en clases y objetos, agrupando datos y comportamientos dentro de entidades independientes que pueden interactuar entre sí.

Principios clave:

- Uso de clases para definir estructuras de datos y sus respectivos métodos.
- Aplicación de conceptos fundamentales como encapsulación, herencia y polimorfismo.
- Enfoque en la organización del software en términos de objetos con atributos y métodos asociados.

Ventajas:

- Modularidad y reutilización: Los objetos pueden ser reutilizados en distintos contextos.

- Mejor organización y mantenimiento: La encapsulación permite estructurar el código de manera clara y eficiente.
- Facilidad de extensión: Es posible agregar funcionalidades sin modificar el código existente.

Desventajas:

- Mayor complejidad inicial: Para problemas simples, definir clases y objetos puede ser innecesario.
- Mayor consumo de memoria: La creación de objetos implica un uso adicional de recursos.
- Mayor cantidad de código: Se requiere definir clases y métodos, lo que aumenta la extensión del programa.

Opinión

Ambos paradigmas presentan características valiosas que los hacen adecuados para distintos escenarios de desarrollo. El paradigma procedural es útil en programas pequeños donde la simplicidad y la eficiencia en el uso de memoria son primordiales. Sin embargo, su falta de organización estructural lo vuelve menos viable para proyectos de gran escala. Por otro lado, el paradigma orientado a objetos ofrece una estructura más robusta y flexible, facilitando la reutilización del código y su mantenimiento. Aunque introduce una mayor complejidad inicial, sus beneficios a largo plazo lo convierten en la opción preferida para el desarrollo de sistemas grandes y escalables. En última instancia, la elección del paradigma depende de la naturaleza del proyecto y los requisitos específicos del software a desarrollar.

Conclusiones

La elección del paradigma de programación adecuado es crucial para el éxito de un proyecto de software. El paradigma procedural es una opción eficiente para proyectos pequeños y tareas específicas, ya que permite una implementación sencilla y un uso optimizado de la memoria. Sin embargo, su falta de modularidad y dificultad para escalar lo hacen poco adecuado para sistemas grandes. Por otro lado, el paradigma orientado a objetos proporciona una estructura más organizada y reutilizable, facilitando la extensión y mantenimiento del código a largo plazo. A pesar de su mayor complejidad y consumo de recursos, su aplicabilidad en sistemas modernos y escalables lo convierte en una alternativa preferida en muchos contextos de desarrollo. En última instancia, la selección del paradigma debe basarse en las necesidades específicas del proyecto, evaluando factores como la escalabilidad, el mantenimiento y la eficiencia en la ejecución.

Apéndice

Paradigma OO

Diagrama de Clases

El siguiente diagrama representa la clase Fraccion y sus atributos y métodos:

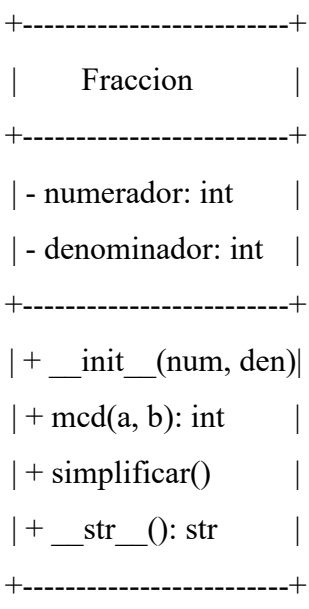
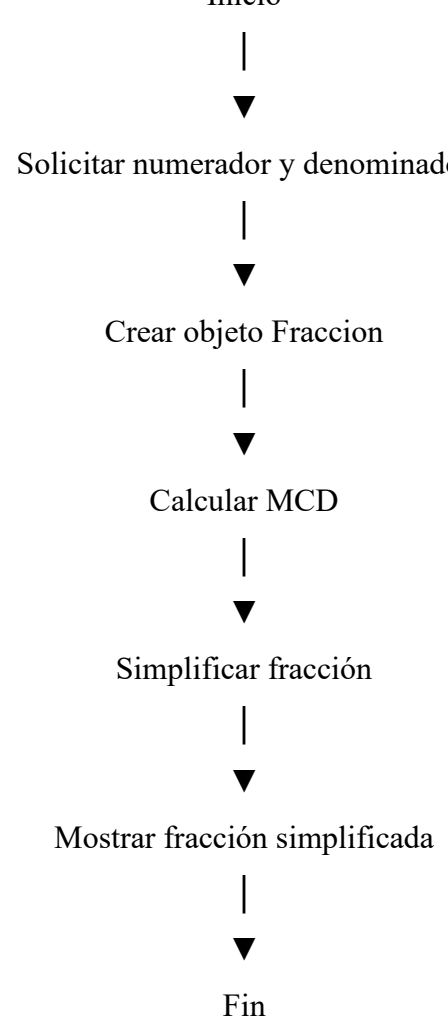


Diagrama de Actividades

El diagrama de actividades muestra el flujo del programa:

1. Inicio
2. Solicitar al usuario el numerador y denominador
3. Crear un objeto de la clase Fraccion
4. Calcular el MCD (Algoritmo de Euclides)
5. Simplificar la fracción dividiendo numerador y denominador por el MCD
6. Mostrar la fracción simplificada
7. Fin



Pseudocódigo

Aquí está el pseudocódigo correspondiente al programa:

```
INICIO
FUNCION MCD(a, b)
    MIENTRAS b ≠ 0 HACER
        temp ← a
        a ← b
        b ← temp MOD b
    FIN MIENTRAS
    RETORNAR a
```

FIN FUNCION

CLASE Fraccion
ATRIBUTOS:
numerador
denominador
METODO __init__(numerador, denominador)
ASIGNAR numerador A self.numerador
ASIGNAR denominador A self.denominador
FIN METODO
METODO simplificar()
divisor ← MCD(self.numerador, self.denominador)
self.numerador ← self.numerador / divisor
self.denominador ← self.denominador / divisor
FIN METODO
METODO __str__()
RETORNAR self.numerador + "/" + self.denominador
FIN METODO
FIN CLASE

FUNCION main()
ESCRIBIR "Ingresa el numerador: "
LEER numerador
ESCRIBIR "Ingresa el denominador: "
LEER denominador
fraccion ← NUEVO Fraccion(numerador, denominador)
fraccion.simplificar()
ESCRIBIR "La fracción simplificada es: " + fraccion
FIN FUNCION

main()
FIN

Programación Estructurada

```python
#Ejemplo para simplificar fracciones paradigma procedural
print("---------PARADIGMA PROCEDURAL----------")



#funcionn que calcula MCD usando el algoritmo de Euclides.
def mcd(a, b):
    while b:
        a, b = b, a % b
    return a



#funcion que simplifica una fracción dividiendo el numerador y el denominador por su MCD
def simplificar_fraccion(numerador, denominador):
    divisor = mcd(numerador, denominador)
    return numerador // divisor, denominador // divisor



 #funcion main
def main():


    # Solicitar al usuario que ingrese la fracción
    numerador = int(input("Ingresa el numerador: "))
    denominador = int(input("Ingresa el denominador: "))



    # Simplificar la fracción
    numerador_simplificado, denominador_simplificado = simplificar_fraccion(numerador, denominador)



    # Mostrar la fracción simplificada
    print(f"La fracción simplificada es: {numerador_simplificado}/{denominador_simplificado}")



#Lo que hace esto es ejecutar el programa llamando a la funcion main
if __name__ == "__main__":
    main()
```

Figura 1. Resolución empleando el Paradigma Procedural

Fuente: Elaboración propia (2025)

Paradigma Procedural

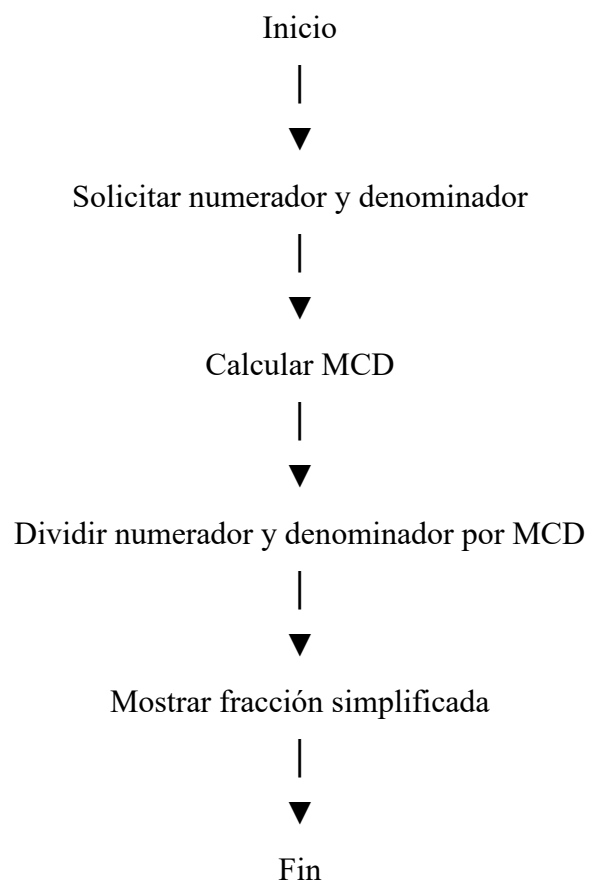
Diagrama de Clases

En este caso, no se realiza un diagrama de clases porque el programa sigue el paradigma procedural, el cual no utiliza clases ni objetos.

Diagrama de Actividades

El diagrama de actividades representa el flujo del programa procedural:

1. Inicio
2. Solicitar al usuario el numerador y denominador
3. Calcular el MCD (Algoritmo de Euclides)
4. Dividir numerador y denominador por el MCD
5. Mostrar la fracción simplificada
6. Fin



Pseudocódigo

El siguiente pseudocódigo representa la lógica del programa:

```
INICIO
FUNCION MCD(a, b)
    MIENTRAS b ≠ 0 HACER
        temp ← a
        a ← b
        b ← temp MOD b
    FIN MIENTRAS
    RETORNAR a
FIN FUNCION

FUNCION simplificar_fraccion(numerador, denominador)
    divisor ← MCD(numerador, denominador)
    RETORNAR (numerador / divisor, denominador / divisor)
FIN FUNCION

FUNCION main()
    ESCRIBIR "Ingresa el numerador: "
    LEER numerador
    ESCRIBIR "Ingresa el denominador: "
    LEER denominador
    (numerador_simplificado, denominador_simplificado) ← simplificar_fraccion(numerador, denominador)
    ESCRIBIR "La fracción simplificada es: " + numerador_simplificado + "/" + denominador_simplificado
```

FIN FUNCION

main()

FIN

Programación Estructurada

```python
#Ejemplo para simplificar fracciones paradigma OO
print("---------PARADIGMA OO----------")


#CLASE FRACCION
class Fraccion:
    #instancia de la clase Fraccion usando los valores ingresados por el usuario
    def __init__(self, numerador, denominador):
        self.numerador = numerador
        self.denominador = denominador



  #METODO que calcula MCD usando el algoritmo de Euclides.
    def mcd(self, a, b):
        while b:
            a, b = b, a % b
        return a



    #METODO que simplifica una fracción dividiendo el numerador y el denominador por su MCD
    def simplificar(self):
        divisor = self.mcd(self.numerador, self.denominador)
        self.numerador //= divisor
        self.denominador //= divisor


    #METODO que devuelve la representación en cadena de la fracción.
    def __str__(self):
        return f"{self.numerador}/{self.denominador}"



#FUNCION principal main
def main():
    # Solicitar al usuario que ingrese la fracción
    numerador = int(input("Ingresa el numerador: "))
    denominador = int(input("Ingresa el denominador: "))

    # Crear un objeto Fraccion

    #fraccion es un objeto de la clase Fraccion
    fraccion = Fraccion(numerador, denominador)

    # Simplificar la fracción
    fraccion.simplificar()

    # Mostrar la fracción simplificada
    print(f"La fracción simplificada es: {fraccion}")



#Lo que hace esto es ejecutar el programa llamando a la funcion main
if __name__ == "__main__":
    main()
```

Figura 2. Resolución Empleando el Paradigma OO

Fuente: Elaboración propia (2025)

Referencias Bibliográficas

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Pressman, R. S. (2014). *Ingeniería del software: un enfoque práctico*. McGraw-Hill.

Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson.

Stroustrup, B. (2013). *The C++ Programming Language* (4th ed.). Addison-Wesley.