UNIVERSIDAD POLITÉCNICA ESTATAL DEL CARCHI

FACULTAD DE INDUSTRIAS AGROPECUARIAS Y CIENCIAS AMBIENTALES

CARRERA DE COMPUTACIÓN

ADMINISTRACIÓN DE BASE DE DATOS

Tema: Paradigma de Programación

Estudiante: Geovanny Basantes

Docente: MSc. Jorge Miranda

Periodo académico: 2025A

Semestre: 7º

Paralelo: AM

Fecha: 09 de marzo de 2025

Tulcán, 2025

Consulta sobre Paradigmas de la Computación

Introducción

La computación ha evolucionado a lo largo del tiempo, adoptando diferentes enfoques y estrategias para resolver problemas. Estos enfoques, conocidos como paradigmas de programación, establecen reglas y principios que definen la manera en que los desarrolladores estructuran y diseñan soluciones computacionales. La comprensión de los distintos paradigmas permiten elegir el más adecuado para una tarea específica, optimizando el rendimiento y la eficiencia de los programas. A continuación, se detallan los principales paradigmas de la computación, junto con sus características, ventajas y desventajas.

Paradigma Imperativo

El paradigma imperativo es el más antiguo y tradicional en la programación. Se basa en la ejecución de instrucciones secuenciales que modifican el estado del sistema paso a paso.

Sus principales características incluyen el uso de algoritmos y procedimientos, variables que almacenan valores y cambian con el tiempo, así como estructuras de control como condicionales e iteraciones.

Entre sus ventajas destaca que facilita el control detallado del flujo del programa, es intuitivo para quienes inician en la programación y tiene un alto rendimiento en términos de eficiencia de recursos. Sin embargo, presenta algunas limitaciones, como la complejidad en programas grandes, lo que puede dificultar su mantenimiento, y la gestión manual de memoria, que puede generar errores como fugas de memoria.

Paradigma Declarativo

A diferencia del imperativo, el paradigma declarativo se centra en describir lo que se desea lograr en lugar de cómo lograrlo.

Sus características incluyen la ausencia de un flujo de control explícito, el uso de reglas y relaciones lógicas, y la interpretación de estas reglas por parte del sistema para

generar los resultados deseados. Ejemplos de lenguajes declarativos incluyen SQL (para bases de datos), Prolog (programación lógica) y XSLT (transformaciones de XML).

Entre sus ventajas se encuentran la reducción de errores, ya que el programador no gestiona manualmente la lógica interna, y un código más conciso y fácil de entender. Sin embargo, puede ser más difícil de optimizar y depende de sistemas que interpreten correctamente las reglas declaradas.

Paradigma Orientado a Objetos (POO)

El paradigma orientado a objetos organiza el código en unidades llamadas "objetos", los cuales contienen datos y métodos para manipular dichos datos.

Entre sus características se incluyen el uso de clases y objetos, los principios clave de encapsulación, herencia y polimorfismo, y la capacidad de modelar sistemas complejos de manera modular. Algunos lenguajes que siguen este paradigma son Java, Python y C++.

Sus ventajas incluyen la facilidad para reutilizar código y la mejora en la organización y mantenimiento del software. No obstante, también presenta ciertas limitaciones, como una mayor complejidad en comparación con los paradigmas imperativos y una ligera sobrecarga de rendimiento debido a la creación y gestión de objetos.

Paradigma Funcional

El paradigma funcional se basa en la aplicación de funciones matemáticas para resolver problemas.

Sus principales características incluyen la evitación del uso de variables mutables, la ausencia de efectos secundarios y el uso de funciones de orden superior. Ejemplos de lenguajes funcionales son Haskell, Lisp y Erlang.

Entre sus ventajas se encuentran la facilidad para implementar concurrencia y paralelismo, así como la reducción de errores al eliminar cambios de estado imprevistos. Sin embargo, puede ser difícil de aprender para programadores con experiencia en paradigmas imperativos y algunos problemas pueden no adaptarse bien a este modelo.

Paradigma Lógico

Este paradigma se basa en reglas lógicas y en la inferencia de hechos a partir de un conjunto de reglas predefinidas.

Sus características incluyen la ausencia de un flujo secuencial tradicional, el uso de hechos y reglas que el sistema emplea para deducir resultados, y su aplicación en inteligencia artificial y sistemas expertos. Ejemplos de lenguajes lógicos son Prolog y Mercury.

Entre sus ventajas se encuentra su alto nivel de abstracción y su utilidad en la resolución de problemas con muchas relaciones lógicas. Sin embargo, su eficiencia en ciertos tipos de problemas es limitada y puede resultar complicado modelar sistemas grandes con este paradigma.

Paradigma Concurrente

Este paradigma está diseñado para sistemas donde se requiere la ejecución simultánea de varias tareas.

Sus características incluyen la gestión de múltiples procesos o hilos de ejecución, el uso de sincronización para evitar conflictos entre hilos y su importancia en sistemas modernos con múltiples núcleos. Algunos lenguajes que permiten la programación concurrente son Java (con hilos), Go (con goroutines) y C# (con TPL y async/await).

Entre sus ventajas destacan la mejora en la eficiencia del hardware multinúcleo y la capacidad de diseñar aplicaciones responsivas. Sin embargo, también introduce problemas como condiciones de carrera y bloqueos, y puede ser difícil de depurar.

Conclusión

Cada paradigma de programación ofrece una forma distinta de abordar problemas computacionales. La elección del paradigma adecuado depende de la naturaleza del problema a resolver, el entorno de ejecución y la experiencia del desarrollador.

En la actualidad, muchos lenguajes modernos combinan varios paradigmas, permitiendo a los programadores seleccionar los enfoques más adecuados según las necesidades de sus proyectos. Comprender los diferentes paradigmas no solo mejora

la capacidad de diseño y desarrollo de software, sino que también amplía las herramientas y metodologías disponibles para resolver problemas de manera eficiente y efectiva.

Referencias:

Trejos Buriticá, O. I. (2014). Relaciones de aprendizaje significativo entre dos paradigmas de programación a partir de dos lenguajes de programación. *Tecnura*, *18*(41), 91-102.

Olabe, X. B., Basogain, M. Á. O., & Basogain, J. C. O. (2015). Pensamiento Computacional a través de la Programación: Paradigma de Aprendizaje. *Revista de educación a distancia (RED)*, (46).

Buriticá, O. I. T. (2017). *Programación imperativa con lenguaje C*. Ecoe Ediciones.