

M8 Modelado orientado a objetos FIN A

Actividad 2

|  |  |
| --- | --- |
| **Tutor:** | **Melissa Gamez Whijares** |
| **Estudiante:** | **José Ramón Ibáñez Posadas** |
| **Matricula:** | **BNL098377** |

|  |  |
| --- | --- |
| San Nicolás de los Garza, Nuevo León | viernes, 29 de agosto de 2025 |

Introducción

En el vasto universo de la programación, la resolución de problemas es una habilidad fundamental que se perfecciona con el conocimiento y la aplicación de diversas herramientas. Las estructuras de control y de datos son el esqueleto de cualquier algoritmo, permitiendo a los desarrolladores organizar la lógica y manipular la información de manera eficiente. Comprender las características de cada una de estas estructuras, como las condicionales y las repetitivas, es crucial para seleccionar la más adecuada para un problema específico. No se trata solo de escribir código que funcione, sino de hacerlo de forma optimizada, legible y mantenible.

La elección correcta de una estructura puede marcar la diferencia entre un programa lento y uno rápido, entre un código difícil de entender y uno claro. Este trabajo explora algunas de las estructuras más comunes en la programación, como las condicionales if y los ciclos while, do-while y for, así como la estructura de datos Array. Además, se aplicará este conocimiento para resolver una situación práctica, desarrollando un programa en Java que calcule el promedio de un grupo de alumnos. A través de este ejercicio, se busca consolidar la comprensión de cómo estas herramientas se complementan para construir soluciones robustas y efectivas en el ámbito del modelado orientado a objetos.

Desarrollo

Resumen de estructuras

Estructura condicional “If”

La estructura if (si) se utiliza para tomar decisiones. Permite ejecutar un bloque de código solo si una condición booleana es verdadera. Es la estructura de control más básica y esencial para la toma de decisiones en un programa.

* **Características Principales:** Evalúa una sola condición. Puede complementarse con else (si no) para ejecutar un bloque alternativo cuando la condición es falsa, y con else if (si no, si) para múltiples condiciones.
* Pasos:

1. El programa llega a la instrucción if.
2. Evalúa la expresión booleana dentro de los paréntesis.
3. Si la expresión es verdadera, ejecuta el bloque de código dentro de las llaves.
4. Si la expresión es falsa, salta el bloque de código y continúa la ejecución.

* **Esquema representativo:**

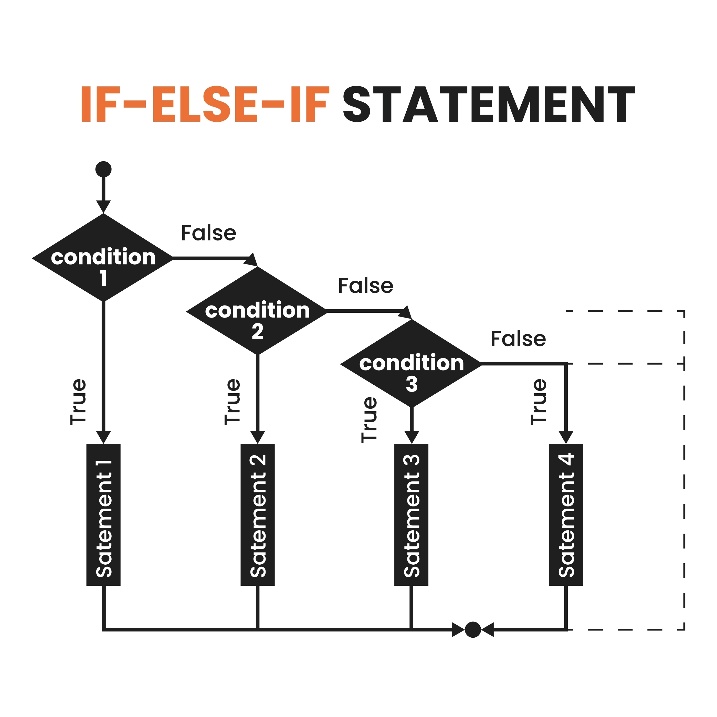


Ilustración 1 - Estructura condicional If

Estructura repetitiva “While”

El ciclo while (mientras) es una estructura de repetición que ejecuta un bloque de código continuamente mientras una condición sea verdadera. La condición se evalúa antes de cada iteración del ciclo.

* **Características Principales:** Ideal para situaciones donde el número de repeticiones es desconocido. El ciclo puede no ejecutarse nunca si la condición inicial es falsa.
* **Pasos:**
  1. El programa evalúa la condición del ciclo.
  2. Si la condición es verdadera, ejecuta el bloque de código.
  3. Al finalizar el bloque, regresa al paso 1 para reevaluar la condición.
  4. Si la condición se vuelve falsa, el ciclo termina.
* **Esquema Representativo:**

A white canvas with arrows and black text

AI-generated content may be incorrect.

Ilustración 2 - Estructura repetitiva while

Estructura repetitiva “Do - While”

El ciclo do-while (hacer-mientras) es similar a while, pero garantiza que el bloque de código se ejecute al menos una vez, ya que la condición se evalúa al final de cada iteración.

* **Características Principales:** La ejecución del bloque de código está garantizada al menos una vez. La condición se verifica después de la primera ejecución.
* **Pasos:**
  1. El programa ejecuta el bloque de código.
  2. Evalúa la condición.
  3. Si la condición es verdadera, regresa al paso 1 para otra iteración.
  4. Si la condición es falsa, el ciclo termina.
* **Esquema Representativo:**

A white board with blue writing

AI-generated content may be incorrect.

Ilustración 3 - Estructura repetitiva do-while

Estructura repetitiva “For”

El ciclo for (para) es una estructura de repetición ideal cuando se conoce de antemano el número de iteraciones.

* **Características Principales:** Sencillo y compacto. Combina la inicialización de la variable de control, la condición de terminación y la expresión de actualización en una sola línea.
* **Pasos:**
  1. Se inicializa la variable de control.
  2. Se evalúa la condición de terminación.
  3. Si la condición es verdadera, se ejecuta el bloque de código.
  4. Se ejecuta la expresión de actualización (incremento o decremento).
  5. Regresa al paso 2 para reevaluar la condición.
  6. Si la condición es falsa, el ciclo termina.
* **Esquema Representativo:**

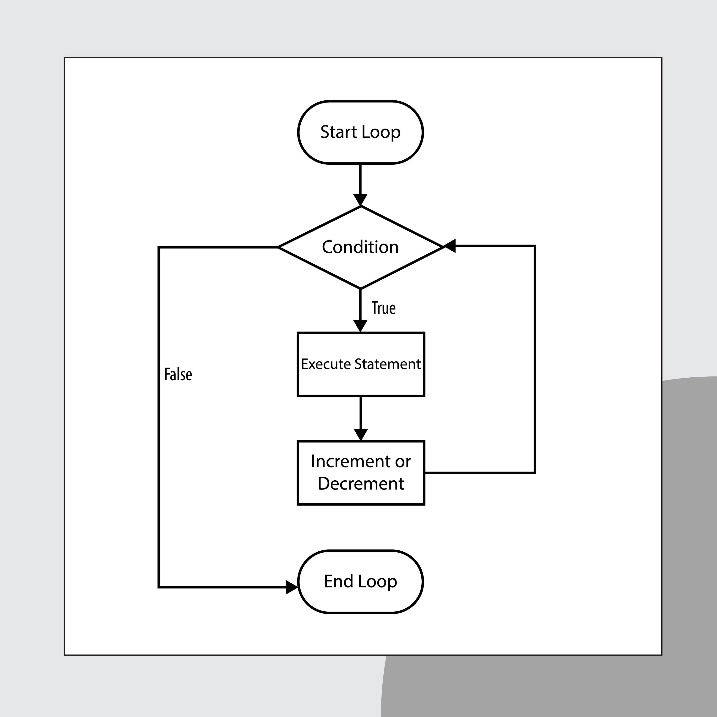


Ilustración 4 - Estructura repetitiva for

Estructura de datos Array

Un Array (arreglo o matriz) es una colección de elementos del mismo tipo de datos, almacenados en posiciones de memoria contiguas. Se accede a los elementos mediante un índice numérico.

* **Características Principales:** Almacena un número fijo de elementos. El acceso a los elementos es muy rápido a través del índice. Todos los elementos deben ser del mismo tipo.
* **Pasos:**
  1. Se declara el arreglo especificando el tipo de datos y el tamaño.
  2. Se inicializan los elementos (opcionalmente).
  3. Se accede a los elementos utilizando un índice, que comienza en cero.
* **Esquema representativo:**

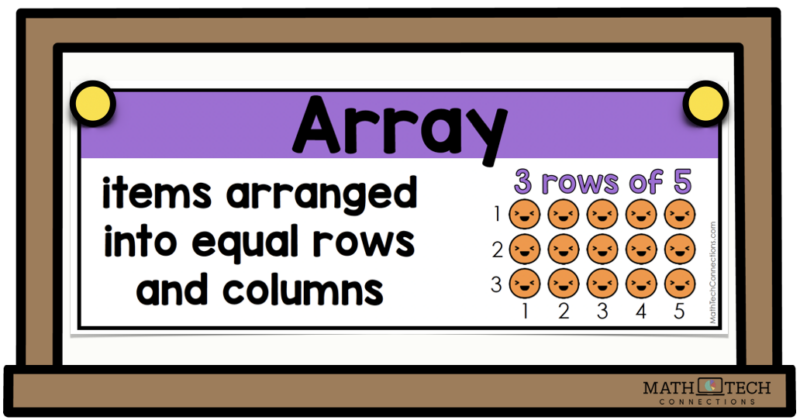


Ilustración 5 - estructura de datos Array

Análisis y programa en BlueJ

Para resolver este problema de manera eficiente, la mejor estructura de control es el ciclo for. Esto se debe a que el problema especifica un número fijo de alumnos (30) a procesar. El ciclo for es ideal para este tipo de situaciones, ya que permite inicializar, controlar y actualizar un contador de manera compacta, garantizando que el programa se repetirá exactamente 30 veces, una para cada alumno.

Dentro del ciclo for, utilizaremos una estructura de control condicional if para determinar si el promedio de cada alumno es mayor o igual a 7 y mostrar el resultado correspondiente (Aprobado o Reprobado).

A screenshot of a computer

AI-generated content may be incorrect.

Ilustración 6 - Calculo promedio

A screenshot of a computer

AI-generated content may be incorrect.

Ilustración 7 - Código fuente clase calculo promedio

A screenshot of a computer

AI-generated content may be incorrect.

Ilustración 8 - ejecución del programa calculo promedio

Diagrama de flujo

A screenshot of a diagram

AI-generated content may be incorrect.

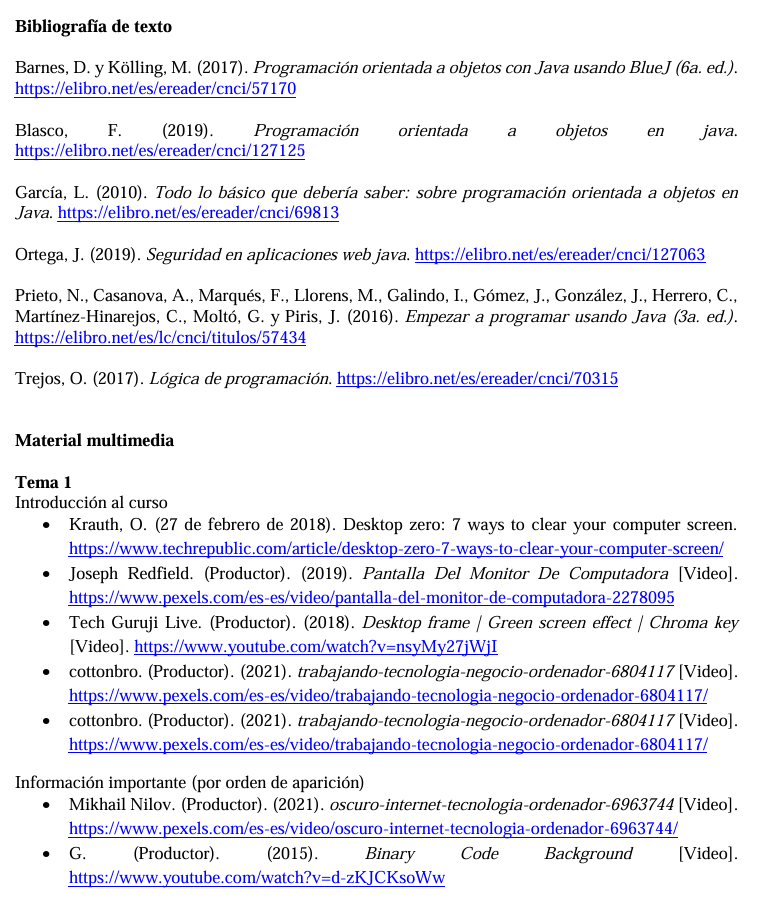
<https://www.mermaidchart.com/app/projects/fdffc541-74d1-49a5-98a5-ce0b004c1d9b/diagrams/45f7b09a-fe5e-40ba-b187-4d393d6d07b7/share/invite/eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkb2N1bWVudElEIjoiNDVmN2IwOWEtZmU1ZS00MGJhLWIxODctNGQzOTNkNmQwN2I3IiwiYWNjZXNzIjoiVmlldyIsImlhdCI6MTc1NjUzMjY5M30.wFnKWFUnbpFwpb1lqd3RFwrEpe6K7sQmtVk0HI4CQlE>

Conclusión

A lo largo de este proyecto, he aprendido que el modelado orientado a objetos no se limita a la creación de clases y objetos, sino que también implica una comprensión profunda de las estructuras de datos y de control. La elección de la estructura correcta es un factor crítico que influye directamente en la eficiencia y la legibilidad del código. En el caso del problema de cálculo de promedios, la estructura for demostró ser la más adecuada debido al número fijo de alumnos, lo que simplificó la lógica y evitó la necesidad de bucles con condiciones más complejas. Este tipo de decisiones de diseño son fundamentales para construir programas escalables y fáciles de mantener.

En el contexto más amplio del almacenamiento de información y las cuestiones financieras, la importancia de estas estructuras se magnifica. Por ejemplo, en una red informática, la manipulación de grandes volúmenes de datos financieros requiere algoritmos eficientes que puedan procesar información de manera rápida y precisa. Un bucle mal diseñado en una aplicación de análisis de datos financieros podría llevar a una carga de procesamiento excesiva, consumiendo recursos de la red y retrasando la entrega de resultados críticos. Por lo tanto, el conocimiento de las estructuras de programación no es solo una habilidad técnica, sino una competencia estratégica para optimizar el rendimiento y garantizar la confiabilidad en sistemas complejos, especialmente en un mundo cada vez más interconectado donde cada milisegundo cuenta.

**Bibliografía**

****