Clase 27/01/2025

Ejercicio 1

```
public class Main {
    public static void main(String[] args) {
        int resultado = 1;

        for (int i = 1; i <= 5; i++) {
            resultado = resultado * 10;
        }


        System.out.printf("%.2f\n",2.55 * resultado);
 }
}
```

El código funciona correctamente en Visual Studio Code configurado para Java, ya que no presenta errores de sintaxis ni de ejecución. Sin embargo, podría surgir un detalle técnico: el uso de “%.2f” en “printf” espera un número en formato flotante, y aunque Java convierte implícitamente el resultado de “2.55 * resultado”, puede haber pérdida de precisión en cálculos con valores muy grandes.

Ejercicio 2

```
public class ErrorAcumulado
{
    public static void main(String[] args)
    {
        float sumaFloat = 0.0f;
        double sumaDouble = 0.0;

        for (int i = 0; i < 10; i++){
            sumaFloat += 0.1f;
            sumaDouble += 0.1;
        }

        System.out.println("Resultado con float: " + sumaFloat);
        System.out.println("Resultado con double: " + sumaDouble);
        System.out.println("Error absoluto con float: " + Math.abs(1.0 -
sumaFloat));
        System.out.println("Error absoluto con double: " + Math.abs(1.0 -
sumaDouble));
    }
}
```

En este ejercicio se puede apreciar un error de precisión. Lo que ocurre es que el tipo de dato float tiene una precisión de aproximadamente 7 número decimales, al sumar 0.1 diez veces para llegar al 1, el sumaFloat muestra un resultado cercano a 1.0 (0.99999994), mientras que el tipo de dato double tiene una mayor precisión (15 a 16 dígitos), por lo que resulta en 1.0. Esto se debe a la representación binaria de los números, que provoca que las sumas acumulativas no sean exactas.

El error absoluto, que es la diferencia entre el resultado obtenido y el esperado (1.0), es más significativo en float que en double.

Ejercicio 3

```
public class Main
{
	public static void main(String[] args)
	{
		int N = 10000;
	  double piReal = Math.PI;
	  double piAprox = 0.0;

	  System.out.printf("%-10s %-15s %-15s %-15s%n", "N", "Pi
	  Aproximado", "Error Absoluto", "Error Relativo");

	  for (int n = 1; n <= N; n++)
	  {

		piAprox += Math.pow(-1, n - 1) / (2.0 * n - 1);

		double piCalc = 4 * piAprox;

				double errorAbsoluto = Math.abs(piReal - piCalc);
		double errorRelativo = errorAbsoluto / piReal;

		if (n % 1000 == 0)
		{
		    System.out.printf("%-10d %-15.10f %-15.10f %-15.10f%n",
		    n, piCalc, errorAbsoluto, errorRelativo);
		}
	    }
	}
}
```

En este ejercicio, se calcula el valor de π utilizando la serie de Leibniz, una aproximación matemática que suma términos alternantes (positivos y negativos).

$$\pi = 4 \cdot \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{2n-1}$$

Se observa que, a medida que se incrementa N (el número de términos), el valor aproximado de π se acerca al valor real. Sin embargo, debido a la naturaleza de la serie, la convergencia es lenta, lo que genera un error absoluto (diferencia entre el valor real y el aproximado). El error relativo, que es el cociente del error absoluto entre el valor real, también disminuye gradualmente.

Esto refleja que el cálculo de π mediante esta serie depende del número de términos utilizados y que la precisión aumenta conforme N crece. Sin embargo, la naturaleza alternante de la serie provoca una convergencia más lenta en comparación con otros métodos de cálculo de π.