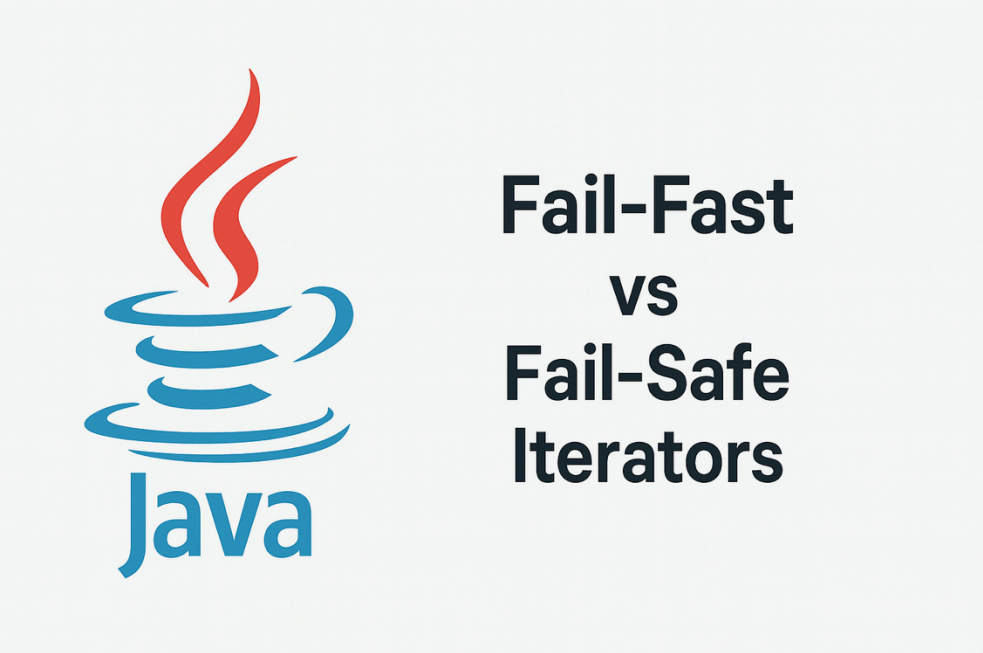
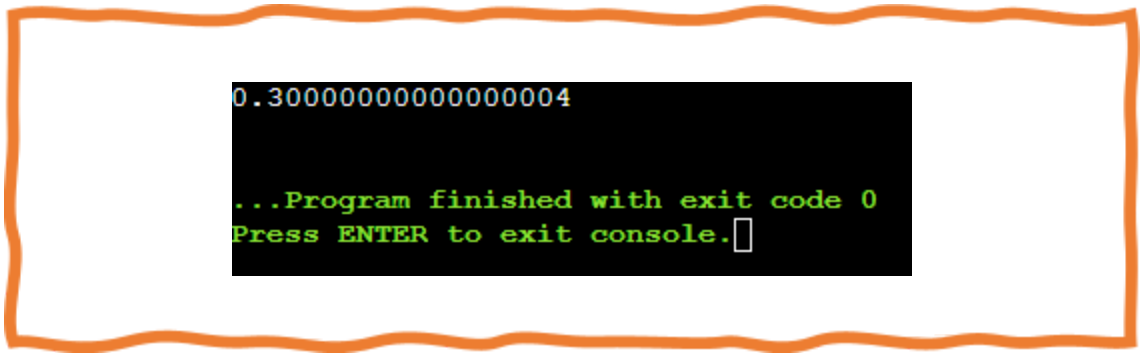
Errores de Programación



1.- Error de Redondeo Binario

El error más común es intentar representar números decimales exactos (como `0.1`) en un sistema binario. Esto causa que operaciones simples devuelvan resultados inesperados.

- **Ejemplo:** `System.out.println(0.1 + 0.2);` imprime `0.30000000000000004` en lugar de `0.3`.
- **Causa:** Fracciones decimales no tienen una representación binaria finita.



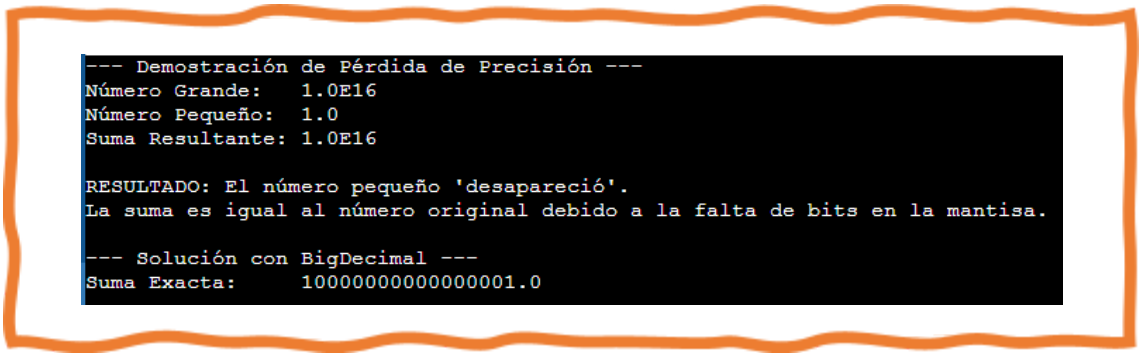
2. Pérdida de Precisión por Magnitud(IEEE 754)

Cuando se operan números con magnitudes muy diferentes, el número más pequeño puede "desaparecer" debido a que no cabe en la mantisa del número más grande.

- **Ejemplo:** Sumar `1.0` a un `double` extremadamente grande puede no cambiar el valor original.

```
public class PerdidaPrecision {
   public static void main(String[] args) {
      // Un double tiene aproximadamente 15-17 dígitos significativos de precisión
      double numeroGrande = 1.0e16; // 1 seguido de 16 ceros
      double numeroPequeno = 1.0;

      double resultado = numeroGrande + numeroPequeno;

      System.out.println("--- Demostración de Pérdida de Precisión ---");
      System.out.println("Número Grande:   " + numeroGrande);
      System.out.println("Número Pequeño:  " + numeroPequeno);
      System.out.println("Suma Resultante: " + resultado);

      // Verificación lógica
      if (resultado == numeroGrande) {
         System.out.println("\nRESULTADO: El número pequeño 'desapareció'.");
         System.out.println("La suma es igual al número original debido a la falta de bits en la
mantisa.");
      }

      // --- SOLUCIÓN USANDO BIGDECIMAL ---
      java.math.BigDecimal bdGrande = new java.math.BigDecimal("1.0e16");
      java.math.BigDecimal bdPequeno = new java.math.BigDecimal("1.0");
      java.math.BigDecimal bdResultado = bdGrande.add(bdPequeno);
```

```
    System.out.println("\n--- Solución con BigDecimal ---");
    System.out.println("Suma Exacta:     " + bdResultado.toPlainString());
  }
}
```



Comparación Directa con ==

Debido a los errores de redondeo mencionados, comparar dos números `double` usando `==` suele fallar.

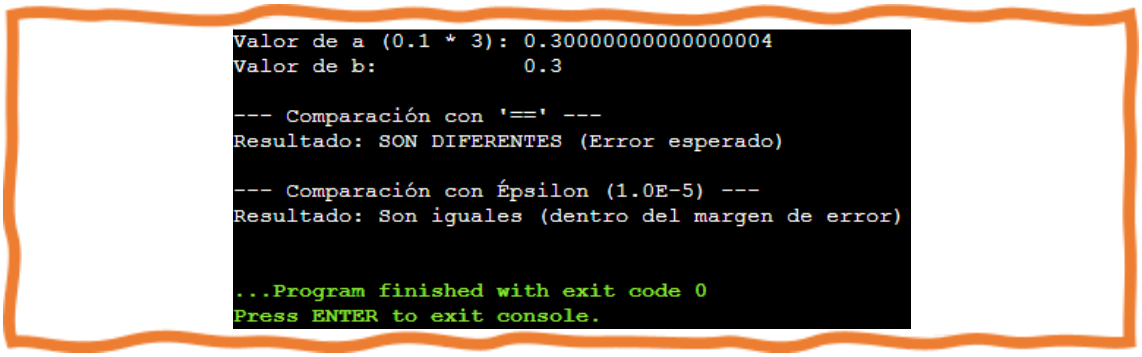
- **Solución:** Se debe usar un margen de error (épsilon).

java

```
if (Math.abs(a - b) < 0.00001) { /* son iguales */ }
```

```
public class ComparacionDouble {
  public static void main(String[] args) {
    // 1. El Problema: Error de redondeo en aritmética de punto flotante
    double a = 0.1 + 0.1 + 0.1;
    double b = 0.3;

    System.out.println("Valor de a (0.1 * 3): " + a);
```

```
        System.out.println("Valor de b:          " + b);

        // 2. Intento de comparación directa (FALLARÁ)
        System.out.println("\n--- Comparación con '==' ---");
        if (a == b) {
            System.out.println("Resultado: Son iguales");
        } else {
            System.out.println("Resultado: SON DIFERENTES (Error esperado)");
        }

        // 3. La Solución: Uso de un margen de error (Épsilon)
        double epsilon = 0.00001; // Definimos la tolerancia

        System.out.println("\n--- Comparación con Épsilon (" + epsilon + ") ---");
        if (Math.abs(a - b) < epsilon) {
            System.out.println("Resultado: Son iguales (dentro del margen de error)");
        } else {
            System.out.println("Resultado: Son diferentes");
        }
    }
}
```



4. Acumulación de Errores en Bucles

Realizar miles de operaciones aritméticas consecutivas con `double` acumula pequeños errores de redondeo que pueden resultar en una desviación significativa al final del proceso.

```
import java.math.BigDecimal;
```

```
public class AcumulacionErroresBucle {
   public static void main(String[] args) {
      int iteraciones = 1000000; // Un millón de sumas
      double incremento = 0.1;

      // 1. Acumulación usando 'double'
      double sumaDouble = 0.0;
      for (int i = 0; i < iteraciones; i++) {
         sumaDouble += incremento;
      }

      // El resultado esperado debería ser exactamente 100,000.0
      double esperado = iteraciones * incremento;

      System.out.println("--- Acumulación en Bucle (1,000,000 de iteraciones) ---");
      System.out.println("Resultado esperado: " + esperado);
      System.out.println("Resultado double:   " + sumaDouble);
      System.out.println("Diferencia (Error): " + (sumaDouble - esperado));

      // 2. Solución usando 'BigDecimal' para precisión absoluta
      BigDecimal sumaBD = BigDecimal.ZERO;
      BigDecimal incrementoBD = new BigDecimal("0.1");

      for (int i = 0; i < iteraciones; i++) {
         sumaBD = sumaBD.add(incrementoBD);
      }

      System.out.println("\n--- Solución con BigDecimal ---");
      System.out.println("Resultado exacto:   " + sumaBD.toPlainString());

      // 3. Verificación de error
      if (sumaDouble != esperado) {
         System.out.println("\nNOTA: El error de 'double' es notable tras un millón de sumas.");
      }
   }
}
```

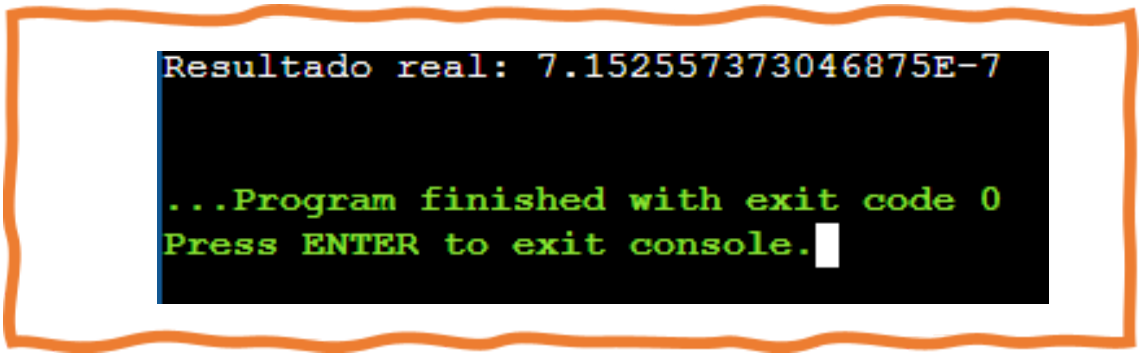
```
Resultado esperado: 100000.0
Resultado double:   100000.00000133288
Diferencia (Error): 1.3328826753422618E-6

--- Solución con BigDecimal ---
Resultado exacto:   100000.0

NOTA: El error de 'double' es notable tras un millón de sumas.


...Program finished with exit code 0
Press ENTER to exit console.
```

Cancelación por Resta (Loss of Significance)

Ocurre cuando restas dos números muy cercanos entre sí. La mayoría de los dígitos significativos se cancelan, dejando solo el error de redondeo como el "resultado" aparente.

java

```java
public class CancelacionPorResta {
    public static void main(String[] args) {
        // Dos números muy grandes y muy cercanos
        double x = 1234567890.1234567;
        double y = 1234567890.1234560;

        // El resultado esperado es 0.0000007
        double resultado = x - y;

        System.out.println("Resultado real: " + resultado);
        // En 2026 notarás que el resultado es 0.0 o un valor basura
        // debido a que los últimos dígitos se perdieron al almacenar x e y.
    }
}
```



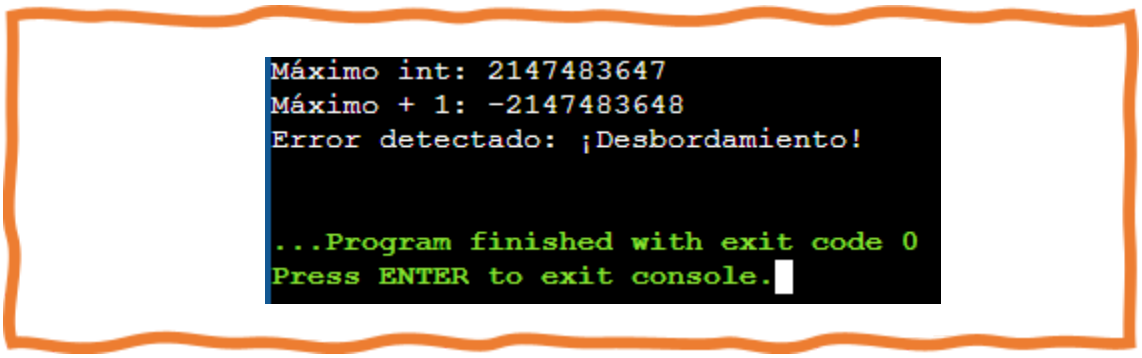
Desbordamiento Silencioso (Overflow)

A diferencia de los errores de decimales, este ocurre con tipos enteros (`int`, `long`). Si superas el valor máximo, Java no lanza una excepción; simplemente "da la vuelta" al número más pequeño (negativo), lo que destruye la exactitud del cálculo.

java

```java
public class DesbordamientoEntero {
    public static void main(String[] args) {
        int max = Integer.MAX_VALUE; // 2,147,483,647
        int resultado = max + 1;

        System.out.println("Máximo int: " + max);
```

```
        System.out.println("Máximo + 1: " + resultado); // Imprime -2147483648

        // Solución: Usar Math.addExact para lanzar una excepción si ocurre
        try {
            Math.addExact(max, 1);
        } catch (ArithmeticException e) {
            System.out.println("Error detectado: ¡Desbordamiento!");
        }
    }
}
```



Conversión Estrecha (Narrowing Primitive Conversion)

Al convertir un tipo de mayor capacidad a uno menor (de `double` a `int` o de `long` a `int`), Java simplemente trunca los bits sobrantes, lo que puede cambiar drásticamente el valor original sin previo aviso.

Java

```
public class ErrorConversion {
    public static void main(String[] args) {
        double valorDouble = 3.14159e10; // Un número muy grande
        int valorInt = (int) valorDouble; // Casting explícito

        System.out.println("Valor Original: " + valorDouble);
        System.out.println("Valor Truncado: " + valorInt);
        // Imprime 2147483647 (el máximo int) porque el double no cabe.
    }
}
```

```
Error: Main method not found in class Main, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application


...Program finished with exit code 1
Press ENTER to exit console.
```

Se salvan los demas lenguajes o APP

R: no

Veamos

EXCEL

La operación `=100,39 - 100,35` puede no dar exactamente `0,04` en la memoria interna de Excel, aunque la celda parezca mostrarlo correctamente.

Pruebalo

Si usas ese resultado en un `=SI(A1=0,04; "OK"; "ERROR")`, podría devolver "ERROR".

INVESTIGAR ERRORES ABSOLUTOS Y RELATIVOS CON LAS FÓRMULAS CÓMO SE CALCULAN

El error absoluto mide cuánto se equivoca una medida respecto al valor real, y el error relativo mide qué tan grande es ese error en comparación con el propio valor real.

Definiciones

- Error absoluto: diferencia (en valor absoluto) entre el valor real y el valor medido.
- Error relativo: cociente entre el error absoluto y el valor real; a veces se expresa en porcentaje.

Fórmulas

Supón que:

- $\mathrm{Vreal}V\mathrm{real}$= valor real o verdadero.
- $\mathrm{Vmed}V\mathrm{med}$= valor medido o aproximado.

Entonces:

- Error absoluto:

$$\mathrm{Ea}=|\mathrm{Vreal}-\mathrm{Vmed}|Ea=|V\mathrm{real}-V\mathrm{med}|$$

- Error relativo (fracción):

$$\mathrm{Er}=\mathrm{EaVreal}/V\mathrm{real}Ea$$

- Error relativo en porcentaje:

$$\mathrm{Er}(\%)=(\mathrm{Ea}/\mathrm{Vreal})\times 100$$

Estas expresiones no llevan unidades en el caso del error relativo; el error absoluto sí tiene las mismas unidades que la medida (cm, kg, s, etc.).

Ejemplo paso a paso

Imagina que la longitud real de una mesa es 2.00 m y tú mides 1.94 m.

1. Calcula el error absoluto:

$$Er = |2.00 - 1.94| = 0.06 \text{ m}$$

1. Calcula el error relativo (fracción):

$$Er = 0.06 / 2.00 = 0.03$$

1. Calcula el error relativo en porcentaje:

$$Er(\%) = 0.03 \times 100 = 3\%$$

Resultado: tu medida es 2.00 m ± 0.06 m, con un error relativo del 3%