

TRABAJO PRÁCTICO 2: PROGRAMACIÓN ESTRUCTURADA

Datos Generales

- **Alumno:** Calcatelli Renzo - rcalcatelli@gmail.com
 - **Comisión:** M2025-1
 - **Materia:** Programación II
 - **Profesor/a:** Ariel Enferrel
 - **Repositorio de Github:** <https://github.com/rcalcatelli/UTN-TUPaD-P2->
-

ÍNDICE

Introducción

1. Estructuras Condicionales

- Ejercicio 1: Verificación de Año Bisiesto
- Ejercicio 2: Mayor de Tres Números
- Ejercicio 3: Clasificación de Edad
- Ejercicio 4: Calculadora de Descuento

2. Estructuras de Repetición

- Ejercicio 5: Suma de Números Pares (while)
- Ejercicio 6: Contador de Positivos y Negativos (for)
- Ejercicio 7: Validación de Nota (do-while)

3. Funciones

- Ejercicio 8: Precio Final con Impuesto
- Ejercicio 9: Costo de Envío
- Ejercicio 10: Actualización de Stock
- Ejercicio 11: Descuento con Variable Global

4. Arrays y Recursividad

- Ejercicio 12: Modificación de Array
- Ejercicio 13: Impresión Recursiva

Conclusiones

Anexos

INTRODUCCIÓN

Este trabajo práctico tiene como objetivo aplicar los conceptos fundamentales de programación estructurada en Java, desde estructuras de control básicas hasta conceptos más avanzados como recursividad. Cada ejercicio está diseñado para reforzar un concepto específico y construir una base sólida de conocimientos.

Los conceptos principales que se abordan son:

- **Estructuras condicionales:** Para toma de decisiones.
 - **Estructuras repetitivas:** Para automatizar procesos.
 - **Funciones:** Para modularizar el código.
 - **Arrays:** Para manejo de colecciones de datos.
 - **Recursividad:** Para soluciones elegantes a problemas complejos.
-

ESTRUCTURAS CONDICIONALES

Las estructuras condicionales nos permiten que nuestro programa tome decisiones basadas en diferentes condiciones. Java nos proporciona `if`, `else if`, `else` y `switch` para estos casos.

Ejercicio 1: Verificación de Año Bisiesto

Concepto aplicado: Operadores lógicos y condiciones compuestas.

Un año bisiesto debe cumplir una regla específica:

- Es divisible por 4, PERO no por 100.
- O es divisible por 400.

Esta lógica requiere el uso de operadores lógicos `&&` (AND) y `||` (OR).

```
Ejercicio1_AñoBisiesto.java
import java.util.Scanner;

Rename usages
public class Ejercicio1_AñoBisiesto {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese un año: ");
        int año = scanner.nextInt();

        // Lógica del año bisiesto:
        // (divisible por 4 Y no divisible por 100) O (divisible por 400)
        boolean esBisiesto = (año % 4 == 0 && año % 100 != 0) || (año % 400 == 0);

        if (esBisiesto) {
            System.out.println("El año " + año + " es bisiesto.");
        } else {
            System.out.println("El año " + año + " no es bisiesto.");
        }

        scanner.close();
    }
}
```

Explicación del código:

- Uso el operador módulo `%` para verificar la divisibilidad.
- La expresión `año % 4 == 0` verifica si es divisible por 4.
- Combino condiciones con `&&` (ambas deben ser verdaderas) y `||` (al menos una debe ser verdadera).

```
Run Ejercicio1_AnioBisiesto x
"C:\Program Files\Java\jdk-21\bin\java.exe"
Ingrese un año: 2024
El año 2024 es bisiesto.
Process finished with exit code 0
```

```
Run Ejercicio1_AnioBisiesto x
"C:\Program Files\Java\jdk-21\bin\java.exe"
Ingrese un año: 1900
El año 1900 no es bisiesto.
Process finished with exit code 0
```

Ejercicio 2: Determinar el Mayor de Tres Números

Concepto aplicado: Comparación secuencial de variables.

```
Ejercicio2_MayorDeTres.java
import java.util.Scanner;

public class Ejercicio2_MayorDeTres { new *
    public static void main(String[] args) { new *
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese el primer número: ");
        int num1 = scanner.nextInt();
        System.out.print("Ingrese el segundo número: ");
        int num2 = scanner.nextInt();
        System.out.print("Ingrese el tercer número: ");
        int num3 = scanner.nextInt();

        // Estrategia: asumir que el primero es el mayor y luego comparar
        int mayor = num1;

        if (num2 > mayor) {
            mayor = num2;
        }
        if (num3 > mayor) {
            mayor = num3;
        }

        System.out.println("El mayor es: " + mayor);

        scanner.close();
    }
}
```

Explicación del algoritmo: Esta es una técnica clásica llamada "búsqueda del máximo". Comenzamos asumiendo que el primer elemento es el mayor, y luego comparamos con los demás elementos, actualizando nuestra variable `mayor` cuando encontramos un valor más grande.

```
Run Ejercicio2_MayorDeTres x
"C:\Program Files\Java\jdk-21\bin\java.exe"
Ingrese el primer número: 8
Ingrese el segundo número: 12
Ingrese el tercer número: 5
El mayor es: 12
Process finished with exit code 0
```

Ejercicio 3: Clasificación de Edad

Concepto aplicado: Cadena de if-else para rangos múltiples.

```
Ejercicio3_ClasificacionEdad.java
import java.util.Scanner;

public class Ejercicio3_ClasificacionEdad {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese su edad: ");
        int edad = scanner.nextInt();

        String clasificacion;

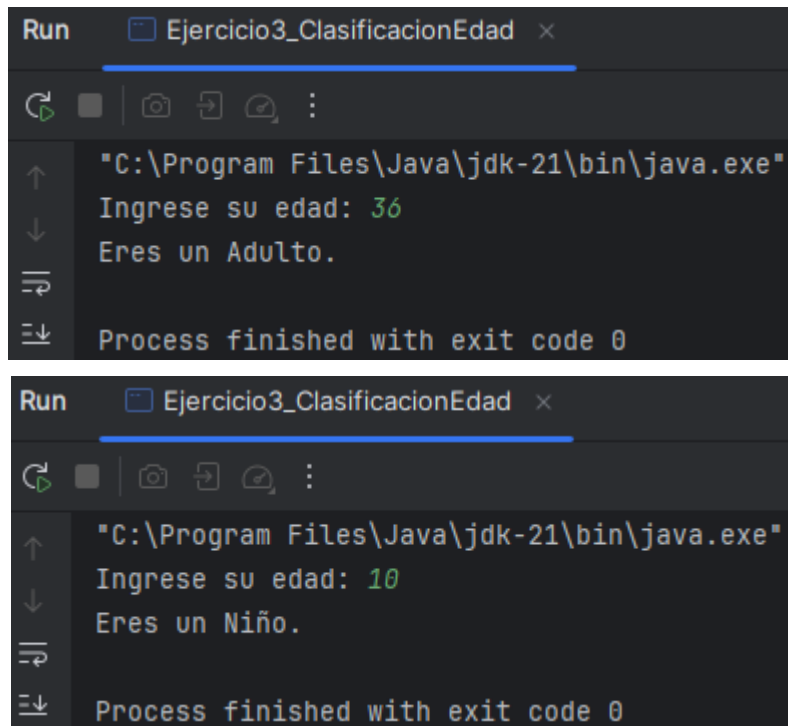
        // Uso if-else encadenado para rangos de edad
        if (edad < 12) {
            clasificacion = "Niño";
        } else if (edad >= 12 && edad <= 17) {
            clasificacion = "Adolescente";
        } else if (edad >= 18 && edad <= 59) {
            clasificacion = "Adulto";
        } else { // 60 años o más
            clasificacion = "Adulto mayor";
        }

        System.out.println("Eres un " + clasificacion + ".");

        scanner.close();
    }
}
```

Explicación de la estructura:

- Uso else if para crear condiciones mutuamente excluyentes.
- Cada rango de edad se evalúa en orden.
- El último else captura todos los casos restantes (60+).



The image shows two screenshots of a Java IDE console window titled "Ejercicio3_ClasificacionEdad".

The first screenshot shows the program execution for an input age of 36. The output is "Eres un Adulto.".

```
"C:\Program Files\Java\jdk-21\bin\java.exe"  
Ingrese su edad: 36  
Eres un Adulto.  
Process finished with exit code 0
```

The second screenshot shows the program execution for an input age of 10. The output is "Eres un Niño.".

```
"C:\Program Files\Java\jdk-21\bin\java.exe"  
Ingrese su edad: 10  
Eres un Niño.  
Process finished with exit code 0
```

Ejercicio 4: Calculadora de Descuento según Categoría

Concepto aplicado: Estructura switch para opciones discretas.

```
Ejercicio4_CalculadoraDescuento.java
import java.util.Scanner;

class Ejercicio4_CalculadoraDescuento { new *
    public static void main(String[] args) { new *
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese el precio del producto: ");
        double precio = scanner.nextDouble();
        System.out.print("Ingrese la categoría del producto (A, B o C): ");
        char categoria = scanner.next().toUpperCase().charAt(0);

        double porcentajeDescuento = 0;

        // Switch es ideal para opciones específicas
        switch (categoria) {
            case 'A':
                porcentajeDescuento = 0.10; // 10%
                break;
            case 'B':
                porcentajeDescuento = 0.15; // 15%
                break;
            case 'C':
                porcentajeDescuento = 0.20; // 20%
                break;
            default:
                System.out.println("Categoría inválida");
                scanner.close();
                return; // Terminar el programa si la categoría es inválida
        }

        double descuentoAplicado = precio * porcentajeDescuento;
        double precioFinal = precio - descuentoAplicado;

        System.out.println("Descuento aplicado: " + (int)(porcentajeDescuento * 100) + "%");
        System.out.println("Precio final: " + precioFinal);

        scanner.close();
    }
}
```

Explicación del switch:

- switch es más legible que múltiples if-else cuando tenemos valores específicos.
- toUpperCase().charAt(0) convierte a mayúscula y toma el primer carácter.
- Cada case debe terminar con break para evitar "fall-through".
- default maneja casos no previstos.


```
Run Ejercicio4_CalculadoraDescuento x
Ingrese el precio del producto: 1000
Ingrese la categoría del producto (A, B o C): B
Descuento aplicado: 15%
Precio final: 850.0
Process finished with exit code 0
```

ESTRUCTURAS DE REPETICIÓN

Las estructuras repetitivas nos permiten ejecutar código múltiples veces. Java nos proporciona tres tipos: `while`, `for` y `do-while`, cada uno apropiado para diferentes situaciones.

Ejercicio 5: Suma de Números Pares (`while`)

Concepto aplicado: Ciclo `while` con condición de parada.

```
Ejercicio5_SumaNumerosPares.java
import java.util.Scanner;

public class Ejercicio5_SumaNumerosPares {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int numero;
        int sumaPares = 0;

        // El ciclo while es perfecto cuando no sabemos cuántas iteraciones necesitamos
        do {
            System.out.print("Ingrese un número (0 para terminar): ");
            numero = scanner.nextInt();

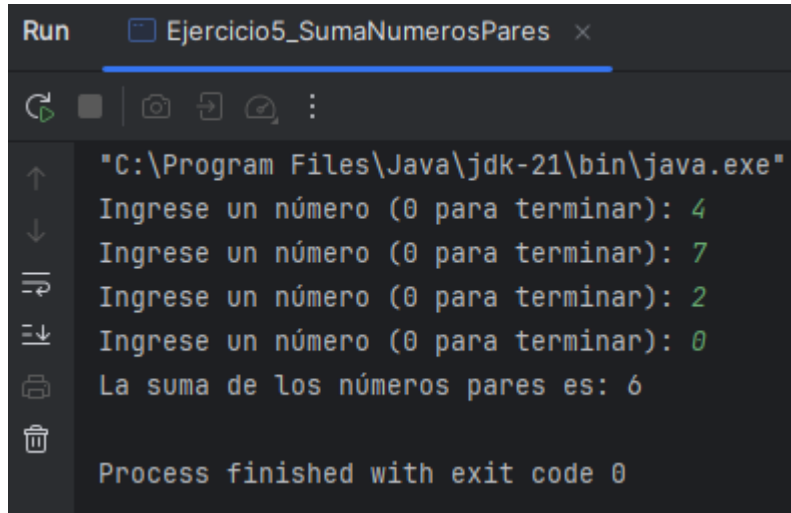
            // Solo sumar si es par y no es cero (el cero técnicamente es par pero es nuestra señal de parada)
            if (numero != 0 && numero % 2 == 0) {
                sumaPares += numero;
            }
        } while (numero != 0);

        System.out.println("La suma de los números pares es: " + sumaPares);

        scanner.close();
    }
}
```

Explicación del algoritmo:

- Uso do-while porque necesito preguntar al menos una vez.
- La condición $\text{numero} \% 2 == 0$ verifica si un número es par.
- El ciclo continúa mientras el número ingresado no sea 0.
- Acumulo la suma en la variable sumaPares .



```
Run Ejercicio5_SumaNumerosPares x
"C:\Program Files\Java\jdk-21\bin\java.exe"
Ingrese un número (0 para terminar): 4
Ingrese un número (0 para terminar): 7
Ingrese un número (0 para terminar): 2
Ingrese un número (0 para terminar): 0
La suma de los números pares es: 6
Process finished with exit code 0
```

Ejercicio 6: Contador de Positivos, Negativos y Ceros (for)

Concepto aplicado: Ciclo for con cantidad fija de iteraciones.

```
Ejercicio6_ContadorNumeros.java
import java.util.Scanner;

public class Ejercicio6_ContadorNumeros { new *
    public static void main(String[] args) { new *
        Scanner scanner = new Scanner(System.in);

        int positivos = 0;
        int negativos = 0;
        int ceros = 0;

        // For es perfecto cuando sabemos exactamente cuántas veces iterar
        for (int i = 1; i <= 10; i++) {
            System.out.print("Ingrese el número " + i + ": ");
            int numero = scanner.nextInt();

            // Clasificar el número según su signo
            if (numero > 0) {
                positivos++;
            } else if (numero < 0) {
                negativos++;
            } else { // numero == 0
                ceros++;
            }
        }

        System.out.println("Resultados:");
        System.out.println("Positivos: " + positivos);
        System.out.println("Negativos: " + negativos);
        System.out.println("Ceros: " + ceros);

        scanner.close();
    }
}
```

Explicación de la estructura for:

- for (int i = 1; i <= 10; i++) ejecuta exactamente 10 iteraciones.
- Uso contadores para cada categoría (positivos++, negativos++, ceros++).
- La estructura if-else if-else clasifica cada número exactamente en una categoría.

```
Run Ejercicio6_ContadorNumeros x
"C:\Program Files\Java\jdk-21\bin\java.exe"
Ingrese el número 1: -5
Ingrese el número 2: 3
Ingrese el número 3: 0
Ingrese el número 4: -1
Ingrese el número 5: 6
Ingrese el número 6: 0
Ingrese el número 7: 9
Ingrese el número 8: -3
Ingrese el número 9: 4
Ingrese el número 10: -8
Resultados:
Positivos: 4
Negativos: 4
Ceros: 2
Process finished with exit code 0
```

Ejercicio 7: Validación de Nota entre 0 y 10 (do-while)

Concepto aplicado: Ciclo do-while para validación de entrada.

```
Ejercicio7_ValidacionNota.java
import java.util.Scanner;

public class Ejercicio7_ValidacionNota { new *
    public static void main(String[] args) { new *
        Scanner scanner = new Scanner(System.in);

        double nota;

        // do-while garantiza que el código se ejecute al menos una vez
        do {
            System.out.print("Ingrese una nota (0-10): ");
            nota = scanner.nextDouble();

            if (nota < 0 || nota > 10) {
                System.out.println("Error: Nota inválida. Ingrese una nota entre 0 y 10.");
            }
        } while (nota < 0 || nota > 10);

        System.out.println("Nota guardada correctamente.");

        scanner.close();
    }
}
```

Explicación del do-while:

- do-while es ideal para validación porque siempre ejecuta el bloque al menos una vez.
- La condición `nota < 0 || nota > 10` se evalúa al final.
- Si la nota es inválida, el ciclo continúa; si es válida, termina.

```
Run Ejercicio7_ValidacionNota x
C:\Program Files\Java\jdk-21\bin\java.exe "-javaage
Ingrese una nota (0-10): 15
Error: Nota inválida. Ingrese una nota entre 0 y 10.
Ingrese una nota (0-10): -2
Error: Nota inválida. Ingrese una nota entre 0 y 10.
Ingrese una nota (0-10): 8
Nota guardada correctamente.

Process finished with exit code 0
```

FUNCIONES

Las funciones nos permiten modularizar el código, hacerlo más legible y reutilizable. Cada función debe tener una responsabilidad específica.

Ejercicio 8: Cálculo del Precio Final con Impuesto y Descuento

Concepto aplicado: Función con parámetros y valor de retorno.

```
Ejercicio8_PrecioFinal.java
import java.util.Scanner;

public class Ejercicio8_PrecioFinal { new *

    // Función que calcula el precio final según la fórmula dada
    public static double calcularPrecioFinal(double precioBase, double impuesto, double descuento) { 1 usage
        // Convertir porcentajes a decimales
        double impuestoDecimal = impuesto / 100;
        double descuentoDecimal = descuento / 100;

        // Aplicar la fórmula: PrecioBase + (PrecioBase * Impuesto) - (PrecioBase * Descuento)
        double precioFinal = precioBase + (precioBase * impuestoDecimal) - (precioBase * descuentoDecimal);

        return precioFinal;
    }

    public static void main(String[] args) { new *
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese el precio base del producto: ");
        double precioBase = scanner.nextDouble();

        System.out.print("Ingrese el impuesto en porcentaje (Ejemplo: 10 para 10%): ");
        double impuesto = scanner.nextDouble();

        System.out.print("Ingrese el descuento en porcentaje (Ejemplo: 5 para 5%): ");
        double descuento = scanner.nextDouble();

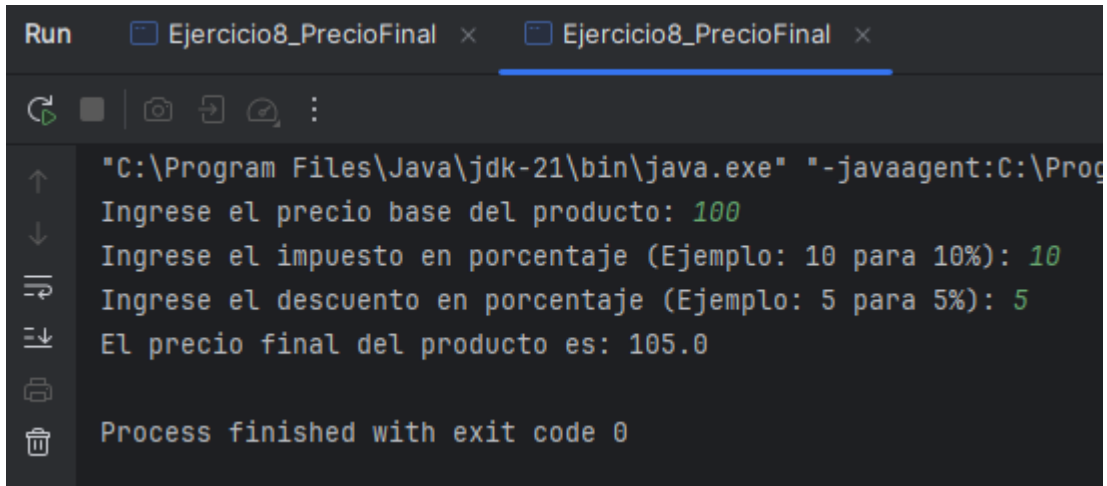
        // Llamar a la función y mostrar el resultado
        double resultado = calcularPrecioFinal(precioBase, impuesto, descuento);

        System.out.println("El precio final del producto es: " + resultado);

        scanner.close();
    }
}
```

Explicación de la función:

- La función recibe tres parámetros y retorna un double.
- Separo la lógica de cálculo del main para mayor claridad.
- La función es reutilizable para cualquier producto.



```
Run  Ejercicio8_PrecioFinal x  Ejercicio8_PrecioFinal x
C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Pro
Ingrese el precio base del producto: 100
Ingrese el impuesto en porcentaje (Ejemplo: 10 para 10%): 10
Ingrese el descuento en porcentaje (Ejemplo: 5 para 5%): 5
El precio final del producto es: 105.0
Process finished with exit code 0
```

Ejercicio 9: Composición de Funciones para Costo de Envío

Concepto aplicado: Múltiples funciones que se llaman entre sí.

```
Ejercicio9_CostoEnvio.java
import java.util.Scanner;

public class Ejercicio9_CostoEnvio { new *

    // Función para calcular el costo de envío según zona y peso
    public static double calcularCostoEnvio(double peso, String zona) { 2 usages new *
        double costoPorKg;

        // Determinar el costo según la zona
        if (zona.equalsIgnoreCase("Nacional")) {
            costoPorKg = 5.0;
        } else if (zona.equalsIgnoreCase("Internacional")) {
            costoPorKg = 10.0;
        } else {
            System.out.println("Zona inválida, usando tarifa nacional.");
            costoPorKg = 5.0;
        }

        return peso * costoPorKg;
    }

    // Función que usa la anterior para calcular el total
    public static double calcularTotalCompra(double precioProducto, double peso, String zona) {
        double costoEnvio = calcularCostoEnvio(peso, zona);
        return precioProducto + costoEnvio;
    }

    public static void main(String[] args) { new *
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese el precio del producto: ");
        double precioProducto = scanner.nextDouble();

        System.out.print("Ingrese el peso del paquete en kg: ");
        double peso = scanner.nextDouble();

        System.out.print("Ingrese la zona de envío (Nacional/Internacional): ");
        String zona = scanner.next();

        // Mostrar el costo de envío por separado
        double costoEnvio = calcularCostoEnvio(peso, zona);
        System.out.println("El costo de envío es: " + costoEnvio);

        // Calcular y mostrar el total
        double total = calcularTotalCompra(precioProducto, peso, zona);
        System.out.println("El total a pagar es: " + total);

        scanner.close();
    }
}
```

Explicación de la composición:

- `calcularCostoEnvio()` maneja la lógica específica del envío.
- `calcularTotalCompra()` usa la función anterior para el cálculo completo.
- Esto demuestra cómo las funciones pueden colaborar entre sí.

```
Run Ejercicio9_CostoEnvio x
C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Progr
Ingrese el precio del producto: 50
Ingrese el peso del paquete en kg: 2
Ingrese la zona de envío (Nacional/Internacional): Nacional
El costo de envío es: 10.0
El total a pagar es: 60.0
Process finished with exit code 0
```

Ejercicio 10: Actualización de Stock

Concepto aplicado: Función con múltiples parámetros para cálculo empresarial.

```
Ejercicio10_ActualizacionStock.java
import java.util.Scanner;

public class Ejercicio10_ActualizacionStock {

    // Función para actualizar el stock según ventas y recepciones
    public static int actualizarStock(int stockActual, int cantidadVendida, int cantidadRecibida) {
        // Fórmula: Stock actual - cantidad vendida + cantidad recibida
        int nuevoStock = stockActual - cantidadVendida + cantidadRecibida;
        return nuevoStock;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese el stock actual del producto: ");
        int stockActual = scanner.nextInt();

        System.out.print("Ingrese la cantidad vendida: ");
        int cantidadVendida = scanner.nextInt();

        System.out.print("Ingrese la cantidad recibida: ");
        int cantidadRecibida = scanner.nextInt();

        // Calcular el nuevo stock
        int stockNuevo = actualizarStock(stockActual, cantidadVendida, cantidadRecibida);

        System.out.println("El nuevo stock del producto es: " + stockNuevo);

        scanner.close();
    }
}
```

Explicación:

- Esta función simula un proceso común en sistemas de inventario.
- Las ventas reducen el stock, las recepciones lo incrementan.
- La función encapsula esta lógica de negocio.

```
Run Ejercicio10_ActualizacionStock x
C:\Program Files\Java\jdk-21\bin\java.exe"
Ingrese el stock actual del producto: 50
Ingrese la cantidad vendida: 20
Ingrese la cantidad recibida: 30
El nuevo stock del producto es: 60
Process finished with exit code 0
```

Ejercicio 11: Descuento Especial con Variable Global

Concepto aplicado: Variables globales vs locales.

```
Ejercicio11_DescuentoEspecial.java
import java.util.Scanner;

public class Ejercicio11_DescuentoEspecial { new *

    // Variable global - accesible desde cualquier método de la clase
    public static final double DESCUENTO_ESPECIAL = 0.10; // 10% 1usage

    // Función que usa la variable global
    public static void calcularDescuentoEspecial(double precio) { 1usage new *
        // Variable local - solo existe dentro de esta función
        double descuentoAplicado = precio * DESCUENTO_ESPECIAL;
        double precioFinal = precio - descuentoAplicado;

        System.out.println("El descuento especial aplicado es: " + descuentoAplicado);
        System.out.println("El precio final con descuento es: " + precioFinal);
    }

    public static void main(String[] args) { new *
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese el precio del producto: ");
        double precio = scanner.nextDouble();

        // Llamar a la función que usa la variable global
        calcularDescuentoEspecial(precio);

        scanner.close();
    }
}
```

Explicación de variables globales vs locales:

- DESCUENTO_ESPECIAL es una variable global (static final).
- descuentoAplicado es una variable local (sólo existe en la función).
- Las variables globales son útiles para constantes que usa toda la aplicación.

```
Run Ejercicio11_DescuentoEspecial x
[C] [X] [O] [D] [V]
"C:\Program Files\Java\jdk-21\bin\java.exe"
Ingresa el precio del producto: 200
El descuento especial aplicado es: 20.0
El precio final con descuento es: 180.0
Process finished with exit code 0
```

ARRAYS Y RECURSIVIDAD

Los arrays nos permiten manejar colecciones de datos, mientras que la recursividad es una técnica donde una función se llama a sí misma.

Ejercicio 12: Modificación de Array de Precios

Concepto aplicado: Manejo básico de arrays y recorridos.

```
Ejercicio12_ArrayPrecios.java

public class Ejercicio12_ArrayPrecios { new *

    public static void main(String[] args) { new *
        // Declarar e inicializar array con precios
        double[] precios = {199.99, 299.5, 149.75, 399.0, 89.99};

        System.out.println("Precios originales:");
        mostrarPrecios(precios);

        // Modificar el precio del tercer producto (índice 2)
        precios[2] = 129.99;

        System.out.println("\nPrecios modificados:");
        mostrarPrecios(precios);
    }

    // Función para mostrar todos los precios usando for-each
    public static void mostrarPrecios(double[] precios) { 2 usages
        for (double precio : precios) {
            System.out.println("Precio: $" + precio);
        }
    }
}
```

Explicación de arrays:

- Los arrays se declaran con `tipo[] nombre = {valores}`.
- Los índices comienzan en 0.
- `for-each` es ideal para recorrer todos los elementos.
- La modificación se hace directamente con `array[indice] = nuevoValor`.

```
Run Ejercicio12_ArrayPrecios x
" C:\Program Files\Java\jdk-21\bin\java.exe"
Precios originales:
Precio: $199.99
Precio: $299.5
Precio: $149.75
Precio: $399.0
Precio: $89.99

Precios modificados:
Precio: $199.99
Precio: $299.5
Precio: $129.99
Precio: $399.0
Precio: $89.99

Process finished with exit code 0
```

Ejercicio 13: Impresión Recursiva de Arrays

Concepto aplicado: Recursividad para recorrido de arrays.

```
Ejercicio13_ArrayRecursivo.java

public class Ejercicio13_ArrayRecursivo { new *

    public static void main(String[] args) { new *
        // Mismo array del ejercicio anterior
        double[] precios = {199.99, 299.5, 149.75, 399.0, 89.99};

        System.out.println("Precios originales:");
        mostrarPreciosRecursivo(precios, 0);

        // Modificar el precio del tercer producto
        precios[2] = 129.99;

        System.out.println("\nPrecios modificados:");
        mostrarPreciosRecursivo(precios, 0);
    }

    // Función recursiva para mostrar precios
    public static void mostrarPreciosRecursivo(double[] precios, int indice) {
        // Caso base: si el índice llegó al final del array, terminar
        if (indice >= precios.length) {
            return;
        }

        // Mostrar el precio actual
        System.out.println("Precio: $" + precios[indice]);

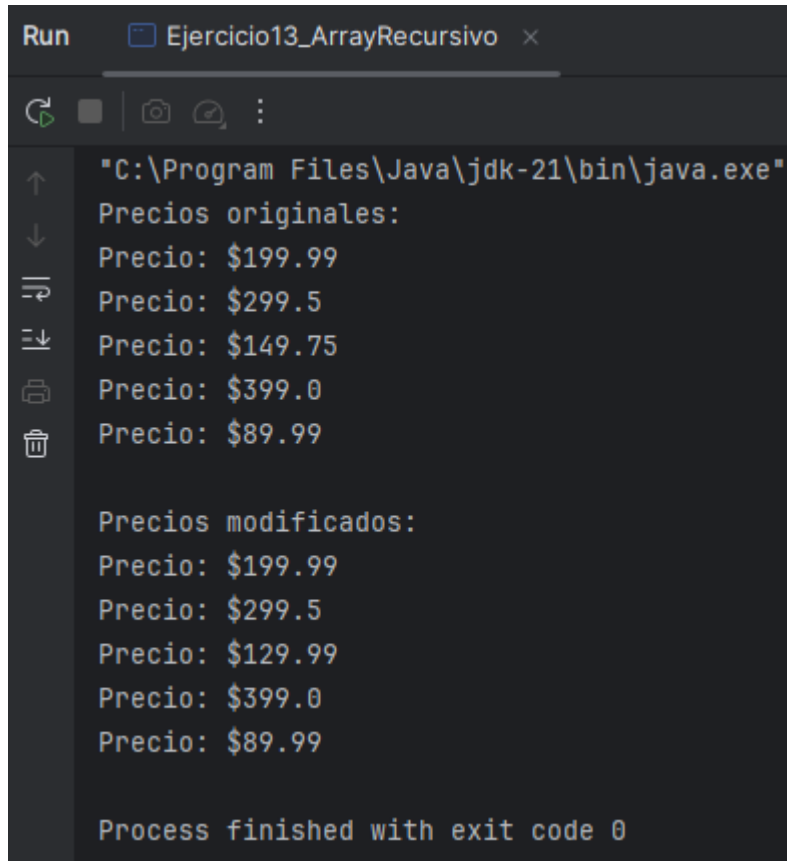
        // Llamada recursiva con el siguiente índice
        mostrarPreciosRecursivo(precios, indice + 1);
    }
}
```

Explicación de la recursividad:

- **Caso base:** `if (indice >= precios.length)` - evita que la recursión sea infinita.
- **Caso recursivo:** `mostrarPreciosRecursivo(precios, indice + 1)` - la función se llama a sí misma.
- Cada llamada procesa un elemento del array y pasa al siguiente.

¿Por qué se usa recursividad en este caso? Aunque un bucle sería más eficiente, la recursividad nos enseña a pensar en términos de:

1. "¿Cuál es el caso más simple?" (array vacío o índice fuera de rango).
2. "¿Cómo reduzco el problema?" (procesar un elemento y pasar al resto).



```
Run Ejercicio13_ArrayRecursivo x
"C:\Program Files\Java\jdk-21\bin\java.exe"
Precios originales:
Precio: $199.99
Precio: $299.5
Precio: $149.75
Precio: $399.0
Precio: $89.99

Precios modificados:
Precio: $199.99
Precio: $299.5
Precio: $129.99
Precio: $399.0
Precio: $89.99

Process finished with exit code 0
```

CONCLUSIONES

A través de este trabajo práctico he aplicado exitosamente los conceptos fundamentales de programación estructurada en Java:

Estructuras de Control Aprendidas:

- **Condicionales:** Implementé decisiones lógicas usando `if-else` y `switch`, aplicando operadores lógicos para condiciones complejas como el año bisiesto.
- **Repetitivas:** Dominé los tres tipos de ciclos: `for` para iteraciones conocidas, `while` para condiciones variables, y `do-while` para validaciones que requieren al menos una ejecución.

Modularización con Funciones:

- Separé la lógica de negocio del código de entrada/salida, creando funciones reutilizables.
- Apliqué composición de funciones, donde unas llaman a otras para resolver problemas complejos.
- Comprendí la diferencia entre variables locales y globales, usando cada una apropiadamente.

Manejo de Datos:

- Trabajé con arrays para almacenar y manipular colecciones de datos.
- Implementé algoritmos de búsqueda y modificación de elementos.
- Comparé enfoques iterativos vs recursivos para el mismo problema.

Recursividad:

- Comprendí el concepto de caso base y caso recursivo.
- Implementé una solución recursiva funcional, aunque simple, que me prepara para problemas más complejos.

Buenas Prácticas Aplicadas:

- Código bien comentado y estructurado
- Nombres de variables descriptivos
- Validación de entrada de usuario
- Separación de responsabilidades en funciones

Este trabajo me ha proporcionado una base sólida para continuar con conceptos más avanzados como programación orientada a objetos, estructuras de datos complejas y algoritmos de mayor sofisticación.

Reflexión Personal:

El ejercicio que más me desafió fue la recursividad, ya que requiere pensar de manera diferente a los bucles tradicionales. Sin embargo, entiendo que es una herramienta poderosa para problemas como árboles, grafos y algoritmos de divide y vencerás.

Los ejercicios de funciones me ayudaron a ver la importancia de escribir código modular y reutilizable, una habilidad fundamental para proyectos más grandes.

ANEXOS

Capturas de Pantalla:

1. **Ejercicio 1:** Ejecución con años 2024, 1900 y 2000.
2. **Ejercicio 2:** Ejecución con números 8, 12, 5.
3. **Ejercicio 3:** Ejecución con edades 25 y 10.
4. **Ejercicio 4:** Ejecución con precio 1000 y categoría B.
5. **Ejercicio 5:** Ejecución con secuencia 4, 7, 2, 0.
6. **Ejercicio 6:** Ejecución con los 10 números del ejemplo.
7. **Ejercicio 7:** Ejecución mostrando validación con valores inválidos.
8. **Ejercicio 8:** Ejecución con precio base 100, impuesto 10%, descuento 5%.
9. **Ejercicio 9:** Ejecución con precio 50, peso 2kg, zona Nacional.
10. **Ejercicio 10:** Ejecución con stock 50, vendidas 20, recibidas 30.
11. **Ejercicio 11:** Ejecución con precio 200.
12. **Ejercicio 12:** Salida mostrando precios originales y modificados.
13. **Ejercicio 13:** Salida idéntica al ejercicio 12 pero con recursividad.