

PROGRAMACIÓN II

Trabajo Práctico 2: Programación Estructurada

OBJETIVO GENERAL

Desarrollar habilidades en programación estructurada en Java, abordando desde conceptos básicos como operadores y estructuras de control hasta temas avanzados como funciones, recursividad y estructuras de datos. Se busca fortalecer la capacidad de análisis y solución de problemas mediante un enfoque práctico,

MARCO TEÓRICO

Concepto	Aplicación en el proyecto
Estructuras condicionales	Clasificación de edad, verificación de año bisiesto
Ciclos (for, while, do-while)	Repetición de ingreso de datos y cálculos
Funciones	Cálculo modular de descuentos, envíos, stock
Arrays	Gestión de precios de productos
Recursividad	Impresión recursiva de arrays

Caso Práctico

Desarrollar los siguientes ejercicios en Java utilizando el paradigma de programación estructurada. Agrupados según el tipo de estructuras o conceptos aplicados:

Estructuras Condicionales:

1. Verificación de Año Bisiesto.

Escribe un programa en Java que solicite al usuario un año y determine si es bisiesto. Un año es bisiesto si es divisible por 4, pero no por 100, salvo que sea divisible por 400.

Ejemplo de entrada/salida:

Ingrese un año: 2024

El año 2024 es bisiesto.

Ingrese un año: 1900

El año 1900 no es bisiesto.

El programa solicita al usuario que ingrese un año y determina si es bisiesto utilizando estructuras de control.

Primero, se lee el año con un “Scanner” y se guarda en la variable “anio”. Luego, mediante una sentencia “if”, se verifica la condición para que un año sea bisiesto. Si se cumple esta condición, el programa imprime que el año es bisiesto, caso contrario, informa que no lo es.

Esta estructura permite resolver el problema de forma clara y directa, aplicando decisiones lógicas básicas sin complejidad adicional.

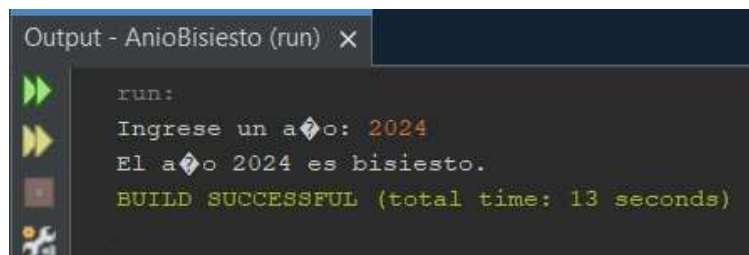
Prueba 1:

Ingreso: 2024

$2024/4 = 506$ ES DIVISIBLE POR 4.

$2024/100 = 20.24$ NO ES DIVISIBLE POR 100.

Cumple ambas condiciones.



```
Output - AnioBisiesto (run) x
run:
Ingrese un año: 2024
El año 2024 es bisiesto.
BUILD SUCCESSFUL (total time: 13 seconds)
```

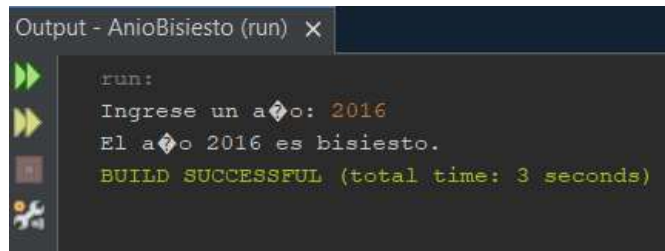
Prueba 2:

Ingreso: 1900

$1900/4=475$ ES DIVISIBLE POR 4

$1900/100=19$ ES DIVISIBLE POR 100.

$1900/400= 4.75$ NO ES DIVISIBLE POR 400.



```
Output - AñoBisiesto (run) x
run:
Ingrese un año: 2016
El año 2016 es bisiesto.
BUILD SUCCESSFUL (total time: 3 seconds)
```

2. Determinar el Mayor de Tres Números.

Escribe un programa en Java que pida al usuario tres números enteros y determine cuál es el mayor.

Ejemplo de entrada/salida:

Ingrese el primer número: 8

Ingrese el segundo número: 12

Ingrese el tercer número: 5

El mayor es: 12

El programa solicita al usuario que ingrese tres números enteros y determina cuál es el mayor.

Primero, se leen los tres números con un “Scanner” y se almacenan en las variables “num1”, “num2” y “num3”.

Luego, mediante la función “esMayor”, se compara cada número: se asume inicialmente que num1 es el mayor, y se actualiza si num2 o num3 son mayores, los va “pisando”.

Finalmente, el programa imprime el mayor de los tres números.

Esta estructura modularizada permite mantener el código limpio y aplicar decisiones lógicas básicas de forma clara y directa.

Prueba 1:

El programa resolvió que el mayor número fue efectivamente el “12” de entre los valores informados.

```
Output - MayorNumero (run) X
run:
Ingrese el primer número: 8
Ingrese el segundo número: 12
Ingrese el tercer número: 5
El mayor de los tres valores ingresados es: 12
BUILD SUCCESSFUL (total time: 9 seconds)
```

3. Clasificación de Edad.

Escribe un programa en Java que solicite al usuario su edad y clasifique su etapa de vida según la siguiente tabla:

Menor de 12 años: "Niño"

Entre 12 y 17 años: "Adolescente"

Entre 18 y 59 años: "Adulto"

60 años o más: "Adulto mayor"

Ejemplo de entrada/salida:

Ingrese su edad: 25

Eres un Adulto.

Ingrese su edad: 10

Eres un Niño.

El programa solicita al usuario que ingrese su edad y clasifica su etapa de vida utilizando estructuras condicionales ("if", "else if", "else").

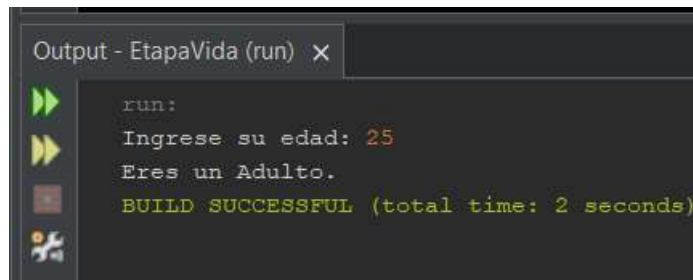
Primero, se lee la edad con un Scanner y se guarda en la variable "edad". Luego, el programa compara la edad según los rangos establecidos:

- Menos de 12 años = Niño
- Entre 12 y 17 = Adolescente
- Entre 18 y 59 = Adulto.
- 60 años o más = Adulto mayor.

Finalmente, imprime en pantalla la etapa correspondiente.

Prueba 1:

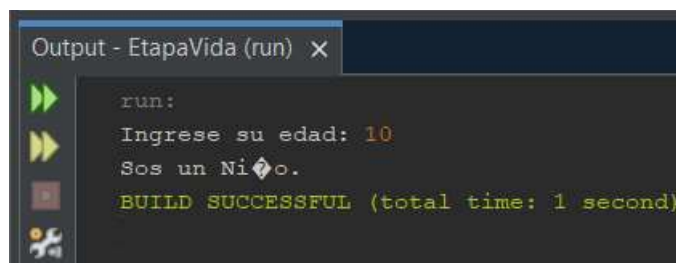
Al ingresar el valor **25** el programa efectivamente devolvió que es "Adulto".



```
Output - EtapaVida (run) X
run:
Ingrese su edad: 25
Eres un Adulto.
BUILD SUCCESSFUL (total time: 2 seconds)
```

Prueba 2:

Al ingresar ahora el valor **10** el programa efectivamente devolvió que es “Niño”.



```
Output - EtapaVida (run) X
run:
Ingrese su edad: 10
Sos un Niño.
BUILD SUCCESSFUL (total time: 1 second)
```

4. Calculadora de Descuento según categoría.

Escribe un programa que solicite al usuario el precio de un producto y su categoría (A, B o C).

Luego, aplique los siguientes descuentos:

Categoría A: 10% de descuento

Categoría B: 15% de descuento

Categoría C: 20% de descuento

El programa debe mostrar el precio original, el descuento aplicado y el precio final

Ejemplo de entrada/salida:

Ingrese el precio del producto: 1000

Ingrese la categoría del producto (A, B o C): B

Descuento aplicado: 15%

Precio final: 850.0

Primero pensé en usar estructuras “if/else if” para determinar el descuento según la categoría (A, B o C). Esto funciona perfectamente, pero luego NetBeans me sugirió usar “switch”, que es más ordenado cuando se tienen varias opciones fijas.

El programa solicita al usuario el precio del producto y su categoría usando un Scanner. Se utiliza "toUpperCase()" para asegurar que la letra se interprete en mayúscula, sin importar cómo la ingrese el usuario.

Luego, mediante un "switch", se determina el porcentaje de descuento:

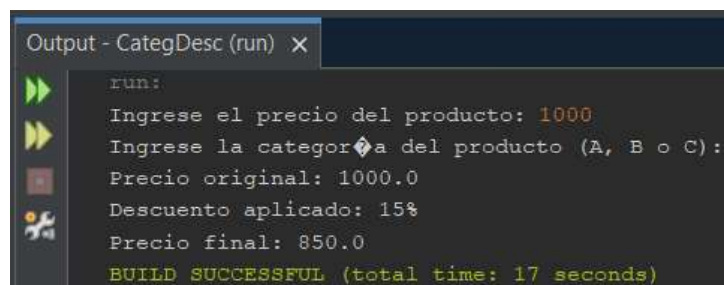
- 10% para A
- 15% para B
- 20% para C

Cada caso termina con break para evitar que se ejecuten los demás.

Finalmente, se calcula el precio final aplicando el descuento y se imprime el precio original, el descuento aplicado y el precio final.

Prueba 1:

Adjunto la prueba donde verifiqué que el programa funciona según lo solicitado en el ejemplo.



```
Output - CategDesc (run) x
run:
Ingrese el precio del producto: 1000
Ingrese la categoría del producto (A, B o C):
Precio original: 1000.0
Descuento aplicado: 15%
Precio final: 850.0
BUILD SUCCESSFUL (total time: 17 seconds)
```

Estructuras de Repetición:

5. Suma de Números Pares (while).

Escribe un programa que solicite números al usuario y sume solo los números pares. El ciclo debe continuar hasta que el usuario ingrese el número 0, momento en el que se debe mostrar la suma total de los pares ingresados.

Ejemplo de entrada/salida:

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

El programa solicita al usuario que ingrese números enteros uno por uno y **suma únicamente los números pares**.

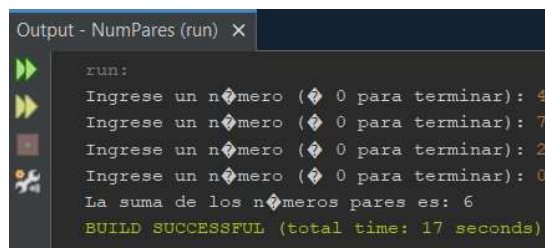
El ciclo de ingreso de números se mantiene usando un “while”, que continúa **hasta que el usuario ingresa el número 0**, donde el programa termina y muestra el resultado.

Dentro del ciclo, se utiliza una estructura condicional “if” para determinar si un número es par mediante el operador **módulo (%)**. Si el número es divisible por 2, se suma a la variable acumuladora “suma”.

Finalmente, cuando el usuario ingresa 0, el ciclo termina y el programa imprime en pantalla la suma total de los números pares ingresados

Prueba 1:

Adjunto la prueba donde verifiqué que el programa funciona según lo solicitado en el ejemplo.



```
run:
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
BUILD SUCCESSFUL (total time: 17 seconds)
```

6. Contador de Positivos, Negativos y Ceros (for).

Escribe un programa que pida al usuario ingresar 10 números enteros y cuente cuántos son positivos, negativos y cuántos son ceros.

Ejemplo de entrada/salida:

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

El programa pide al usuario 10 números enteros usando un bucle “for”.

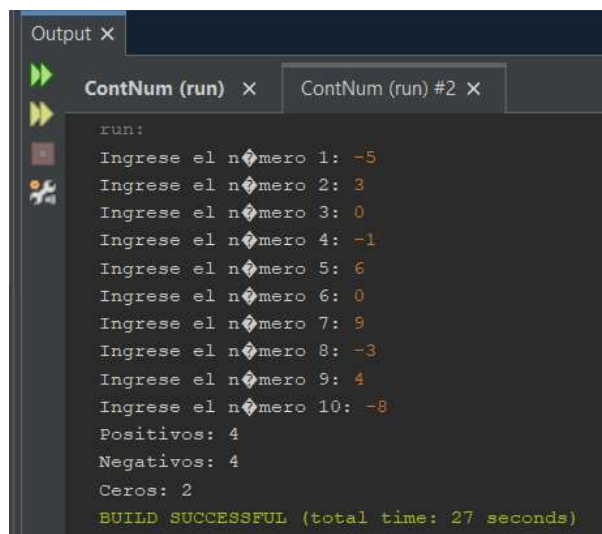
Para cada número ingresado, utiliza condicionales (“If”, “else if”, “else”) para determinar si es **positivo**, **negativo** o **cero** y a su vez acumula la cantidad en las variables correspondientes.

Al finalizar el bucle, imprime el total por pantalla.

Esto permite practicar estructuras de repetición y condicionales, y mantener el código simple y claro.

Prueba 1:

Adjunto la prueba donde verifiqué que el programa funciona según lo solicitado en el ejemplo.



```
Output X
ContNum (run) X ContNum (run) #2 X
run:
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
Positivos: 4
Negativos: 4
Ceros: 2
BUILD SUCCESSFUL (total time: 27 seconds)
```

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

Escribe un programa que solicite al usuario una nota entre 0 y 10. Si el usuario ingresa un número fuera de este rango, debe seguir pidiéndole la nota hasta que ingrese un valor válido.

Ejemplo de entrada/salida:

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.

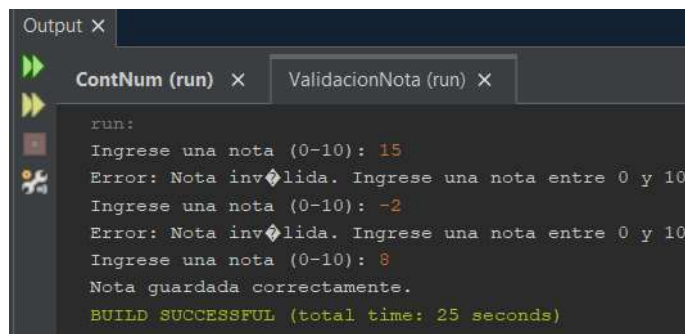
El programa solicita al usuario ingresar una nota entre 0 y 10.

Usa un bucle “do-while”, para asegurarse que el mensaje se muestre al menos una vez. Dentro del bucle, verifica con un condicional “if” si la nota está fuera del rango. Si es así, muestra un mensaje de error y solicita una nueva nota.

Cuando el usuario ingresa un valor válido, el programa termina y confirma que la nota fue guardada con éxito.

Prueba 1:

Adjunto la prueba donde verifiqué que el programa funciona según lo solicitado en el ejemplo ya que regresa el mensaje esperado.



```
run:
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.
BUILD SUCCESSFUL (total time: 25 seconds)
```

Funciones:

8. Cálculo del Precio Final con impuesto y descuento.

Crea un método **calcularPrecioFinal(double impuesto, double descuento)** que calcule el precio final de un producto en un e-commerce. La fórmula es:

$$\text{PrecioFinal} = \text{PrecioBase} + (\text{PrecioBase} \times \text{Impuesto}) - (\text{PrecioBase} \times \text{Descuento})$$
$$\text{PrecioFinal} = \text{PrecioBase} + (\text{PrecioBase} \times \text{Impuesto}) - (\text{PrecioBase} \times \text{Descuento})$$

Desde main(), solicita el precio base del producto, el porcentaje de impuesto y el porcentaje de descuento, llama al método y muestra el precio final.

Ejemplo de entrada/salida:

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

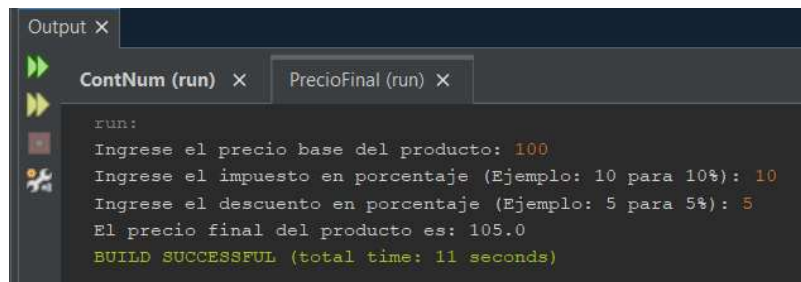
El programa solicita al usuario precio **base**, **porcentaje de impuesto** y **porcentaje de descuento**. El método "calcularPrecioFinal" recibe estos tres valores como parámetros y aplica la fórmula:

$$\text{PrecioFinal} = \text{PrecioBase} + (\text{PrecioBase} \times \text{Impuesto}) - (\text{PrecioBase} \times \text{Descuento}).$$

Desde "main()", se llama al método con los valores ingresados y se muestra el precio final.

Prueba 1:

Según los datos solicitados para realizar la prueba, ésta devolvió exitosamente un correcto valor.



```
Output X
ContNum (run) X PrecioFinal (run) X
run:
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
BUILD SUCCESSFUL (total time: 11 seconds)
```

9. Composición de funciones para calcular costo de envío y total de compra.

a. **calcularCostoEnvio(double peso, String zona)**: Calcula el costo de envío basado en la zona de envío (Nacional o Internacional) y el peso del paquete.

Nacional: \$5 por kg

Internacional: \$10 por kg

b. **calcularTotalCompra(double precioProducto, double costoEnvio)**: Usa **calcularCostoEnvio** para sumar el costo del producto con el costo de envío.

Desde **main()**, solicita el peso del paquete, la zona de envío y el precio del producto. Luego, muestra el total a pagar.

Ejemplo de entrada/salida:

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

El programa solicita al usuario **precio del producto, peso del paquete y zona de envío.**

Se utiliza la función “calcularCostoEnvio” para determinar el costo del envío según la zona:

- **Nacional:** \$5 por kg
- **Internacional:** \$10 por kg

Para comparar la zona ingresada, se usa “equalsIgnoreCase”, lo que permite reconocer la palabra sin importar si el usuario escribe mayúsculas o minúsculas.

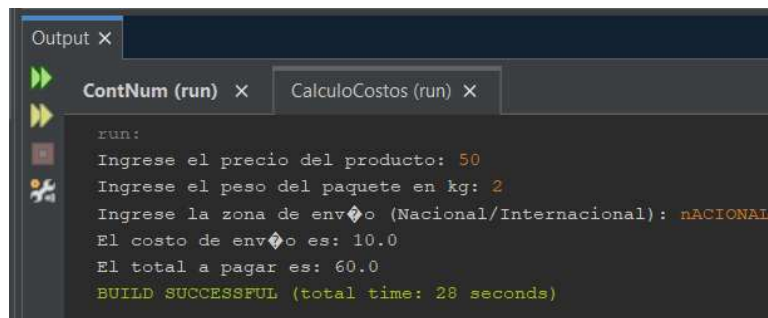
Luego, “calcularTotalCompra” suma el **precio del producto más el costo de envío** para obtener el total a pagar.

Desde main(), se llaman ambos métodos con los valores ingresados por el usuario y se muestran los resultados.

Esto permite **modularizar el código**, usando **funciones simples** para separar cálculos y mantener el programa limpio y entendible.

Prueba 1:

Adjunto la prueba donde verifiqué que el programa funciona según lo solicitado en el ejemplo ya que regresa el total esperado. Probé una manera diferente de informar “Nacional” para comprobar que el método **equalsIgnoreCase** cumple con su deber.



```
Output X
ContNum (run) X  CalculoCostos (run) X
run:
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
BUILD SUCCESSFUL (total time: 28 seconds)
```

10. Actualización de stock a partir de venta y recepción de productos. Crea un método **actualizarStock(int stockActual, int cantidadVendida, int cantidadRecibida)**, que calcule el nuevo stock después de una venta y recepción de productos:

NuevoStock = StockActual – CantidadVendida + CantidadRecibida

NuevoStock = CantidadVendida + CantidadRecibida

Desde **main()**, solicita al usuario el stock actual, la cantidad vendida y la cantidad recibida, y muestra el stock actualizado.

Ejemplo de entrada/salida:

Ingrese el stock actual del producto: 50

Ingrese la cantidad vendida: 20

Ingrese la cantidad recibida: 30

El nuevo stock del producto es: 60

El programa solicita al usuario **stock actual, cantidad vendida y cantidad recibida**.

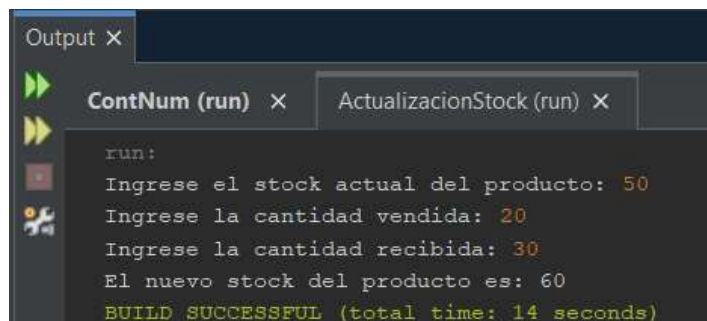
Se utiliza la función “actualizarStock” que recibe esos tres valores como parámetros y calcula el nuevo stock con la fórmula: **NuevoStock = StockActual – CantidadVendida + CantidadRecibida**

Desde “main()”, se llama a “actualizarStock” con los valores ingresados por el usuario y se muestra el resultado por pantalla.

De esta forma, el cálculo del stock se hace de manera clara, sin mezclarlo con lo que el usuario escribe, y el programa se mantiene simple y fácil de seguir.

Prueba 1:

Probando el programa con los valores solicitados en el ejemplo, vemos como resuelve de manera correcta.



```
run:
Ingrese el stock actual del producto: 50
Ingrese la cantidad vendida: 20
Ingrese la cantidad recibida: 30
El nuevo stock del producto es: 60
BUILD SUCCESSFUL (total time: 14 seconds)
```

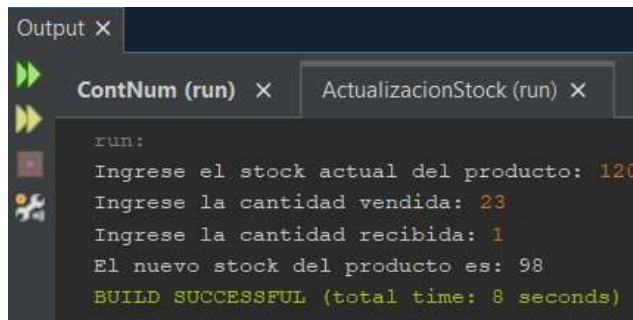
Prueba 2:

Ingrese el stock actual del producto: 120

Ingrese la cantidad vendida: 23

Ingrese la cantidad recibida: 1

El nuevo stock del producto es: 98



```
Output X
ContNum (run) X ActualizacionStock (run) X
run:
Ingrese el stock actual del producto: 120
Ingrese la cantidad vendida: 23
Ingrese la cantidad recibida: 1
El nuevo stock del producto es: 98
BUILD SUCCESSFUL (total time: 8 seconds)
```

11. Cálculo de descuento especial usando variable global.

Declara una variable global **Ejemplo de entrada/salida:** = 0.10. Luego, crea un método **calcularDescuentoEspecial(double precio)** que use la variable global para calcular el descuento especial del 10%.

Dentro del método, declara una variable local **descuentoAplicado**, almacena el valor del descuento y muestra el precio final con descuento.

Ejemplo de entrada/salida:

Ingrese el precio del producto: 200

El descuento especial aplicado es: 20.0

El precio final con descuento es: 180.0

El programa solicita al usuario el precio del producto. Luego, se declara una variable global llamada “descuentoGlobal” con el valor fijo de 10% de descuento. Por eso, el código tiene un comentario para aclarar lo que representa esa variable (**static double descuentoGlobal = 0.10; // 10% de descuento**).

La función “calcularDescuentoEspecial” recibe el precio como parámetro y calcula:

- El descuento aplicado multiplicando el precio por la variable global.
- El precio final restando el descuento al precio original.

Desde “main()”, se llama a la función con el precio ingresado y se muestran ambos valores en pantalla.

Esto permite separar el cálculo del descuento de la interacción con el usuario, usar una variable global para mantener el porcentaje de descuento siempre disponible y mantener el código claro y simple.

Prueba 1:

Adjunto la prueba solicitada verificando los resultados correctos:

```
Output - DescEspecial (run) X
run:
Ingrese el precio del producto: 200
El descuento especial aplicado es: 20.0
El precio final con descuento es: 180.0
BUILD SUCCESSFUL (total time: 10 seconds)
```

Arrays y Recursividad:

12. Modificación de un array de precios y visualización de resultados.

Crea un programa que:

- Declare e inicialice un array con los precios de algunos productos.
- Muestre los valores originales de los precios.
- Modifique el precio de un producto específico.
- Muestre los valores modificados.

Salida esperada:

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

Conceptos Clave Aplicados:

- ✓ Uso de arrays (double[]) para almacenar valores.
- ✓ Recorrido del array con for-each para mostrar valores.
- ✓ Modificación de un valor en un array mediante un índice.
- ✓ Reimpresión del array después de la modificación.

El programa declara un **array precios** con valores de algunos productos.

Se muestran los precios originales usando un bucle “for-each” para recorrer el array.

Luego, se modifica un precio específico usando su índice (**precios[2] = 129.99**).

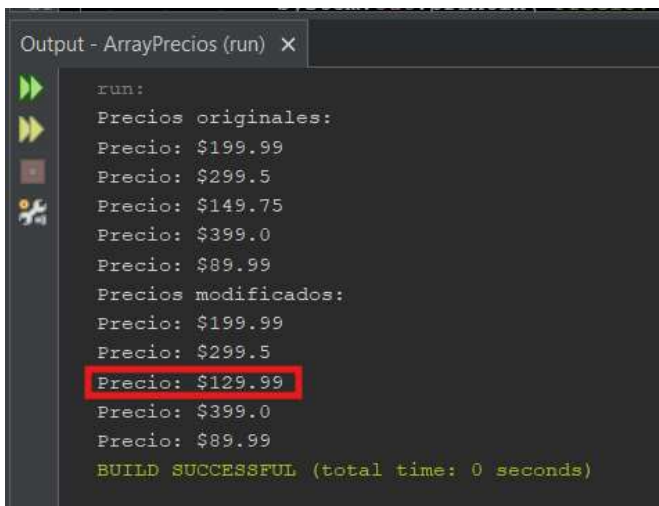
Finalmente, se muestran los precios modificados con otro for-each.

Esto permite:

- Almacenar múltiples valores en un solo array “(double[])”.
- Recorrer y mostrar los valores de manera sencilla con “for-each”.
- Modificar un valor específico usando el índice y volver a mostrar todo el array.

Prueba 1:

Se muestra el mensaje por pantalla que devolvió el código con el reemplazo de precio.



```
Output - ArrayPrecios (run) x
run:
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
BUILD SUCCESSFUL (total time: 0 seconds)
```

13. Impresión recursiva de arrays antes y después de modificar un elemento.

Crea un programa que:

- Declare e inicialice un array con los precios de algunos productos.
- Use una función recursiva para mostrar los precios originales.
- Modifique el precio de un producto específico.

d. Use otra función recursiva para mostrar los valores modificados.

Salida esperada:

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

Conceptos Clave Aplicados:

- ✓ Uso de arrays (`double[]`) para almacenar valores.
- ✓ Recorrido del array con una función recursiva en lugar de un bucle.
- ✓ Modificación de un valor en un array mediante un índice.
- ✓ Uso de un índice como parámetro en la recursión para recorrer el array.

El programa declara un array `precios` con valores de algunos productos.

Se utiliza la función “`mostrarPrecios`” que recibe el array y un índice. La función imprime el valor del índice actual. Luego se llama a sí misma aumentando el índice en 1.

El caso base es cuando el índice llega al tamaño del array (**`indice >= precios.length`**), momento en el cual la función deja de llamarse y la recursión termina.

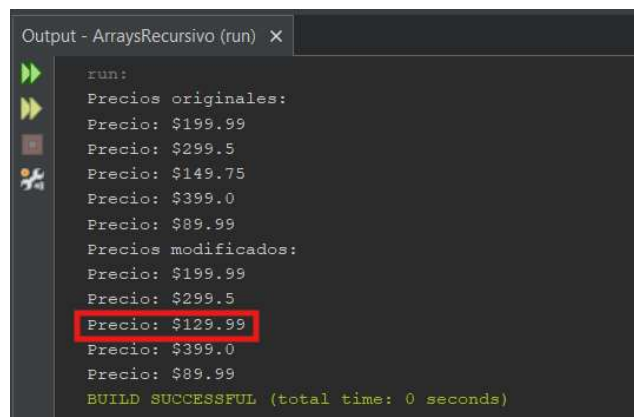
Primero, se muestran los precios originales, luego se modifica un valor específico y se vuelven a mostrar los precios modificados usando la misma función recursiva.

Esto permite:

- Recorrer el array sin usar bucles tradicionales.
- Reutilizar la función para cualquier array de precios.
- Mantener el código claro y ordenado, mostrando la potencia de la recursión en casos simples.

Prueba 1:

Se muestra el mensaje por pantalla que devolvió el código con el reemplazo de precio.



```
Output - ArraysRecursivo (run) X
run:
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
BUILD SUCCESSFUL (total time: 0 seconds)
```

CONCLUSIONES ESPERADAS

- Aplicar estructuras de control y decisión para resolver problemas.
- Diseñar soluciones usando estructuras iterativas y condicionales.
- Modularizar el código utilizando funciones con y sin retorno.
- Utilizar arrays para almacenamiento y manipulación de datos.
- Comprender y aplicar la recursividad en casos simples.
- Trabajar con variables locales y globales de forma adecuada.
- Fortalecer la capacidad de análisis lógico y la resolución de errores.
- Consolidar el uso del lenguaje Java mediante la práctica estructurada.

CONCLUSION FINAL:

Durante este trabajo práctico se aplicaron estructuras de control y decisión para resolver distintos problemas, usando **if/else** y **switch** de manera efectiva.

Se diseñaron soluciones con estructuras repetitivas como **for**, **while** y **do-while**, permitiendo repetir acciones y controlar el flujo de los programas según las condiciones.

Se trabajó con funciones con y sin retorno, lo que **permitió modularizar el código**, separar la lógica de los cálculos de la interacción con el usuario y mantener el código limpio y ordenado.

Se utilizaron **arrays** para almacenar y manipular datos, incluyendo la modificación de elementos y la visualización mediante **for-each** y **recursión**, fortaleciendo la comprensión de cómo recorrer y gestionar colecciones de datos.

Se aplicó la recursividad en casos simples, como mostrar los valores de un array de forma ordenada, y se aprendió a identificar el **caso base**, que detiene la recursión de manera segura.

Se trabajó con variables locales y globales, como en el cálculo de descuentos especiales, entendiendo cómo afectan al alcance y al funcionamiento de los programas.

Todo esto contribuyó a fortalecer la capacidad de análisis lógico y resolución de errores, y a consolidar el uso del lenguaje Java, practicando de manera estructurada conceptos fundamentales para la programación.

En conjunto, este TP permitió desarrollar programas simples, claros y funcionales, sentando una base sólida para continuar aprendiendo y aplicando programación en Java.