



---

# TRABAJO PRÁCTICO 2: PROGRAMACIÓN ESTRUCTURADA

---



# Resolución de los ejercicios:

## 1. Verificación de Año Bisiesto.

**Objetivo:** Implementar un programa que determine si un año entero es bisiesto.

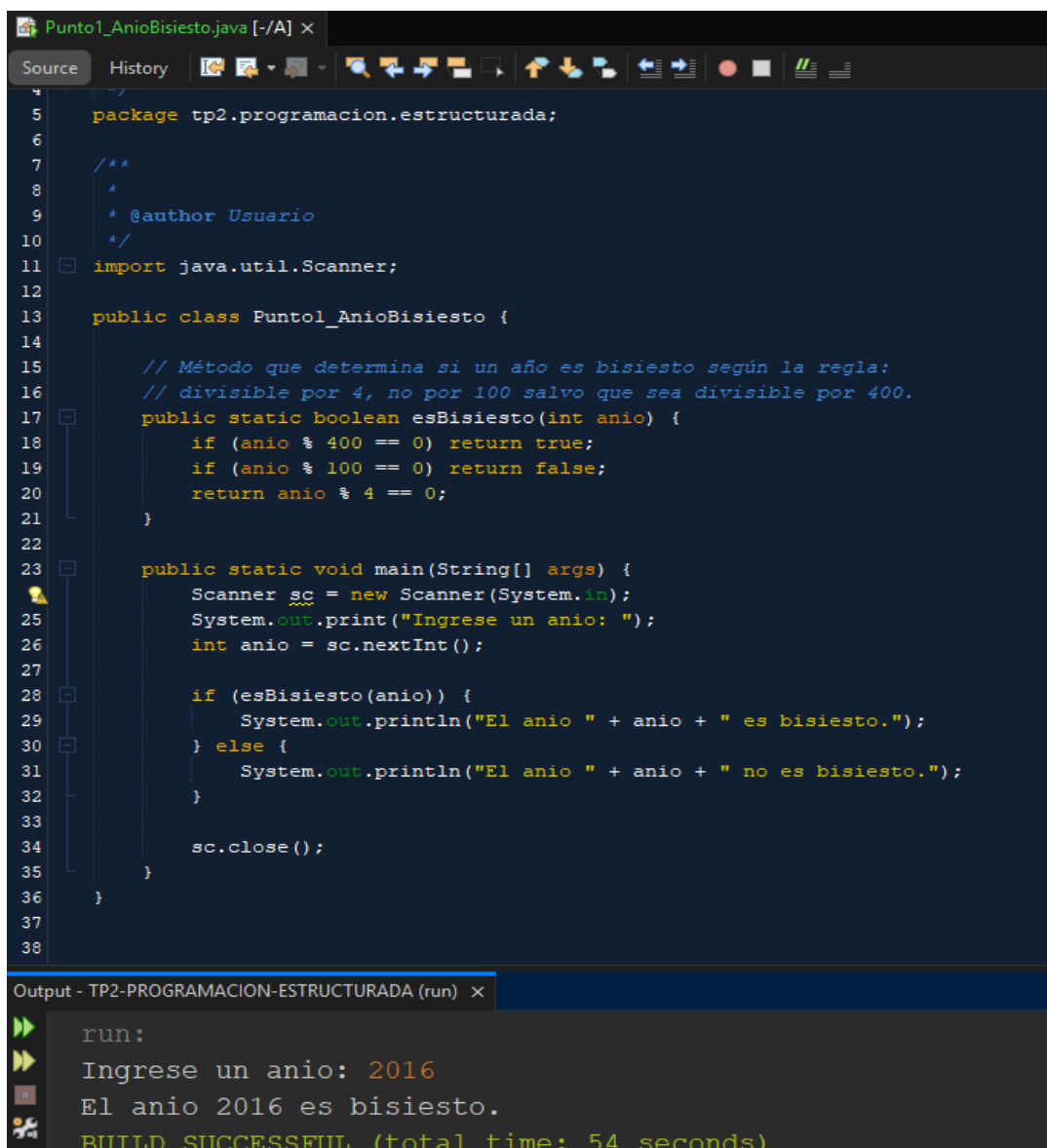
**Algoritmo:** Se verifica primero divisibilidad por 400 (bisiesto), luego por 100 (no bisiesto), luego por 4 (bisiesto). Esta orden evita clasificar mal años como 1900.

**Entrada:** Un entero (año).

**Salida:** Mensaje indicando si el año es bisiesto o no.

**Notas:** Implementado con método esBisiesto(int) para facilitar pruebas unitarias y reutilización. Ejemplos incluidos: 2024 (sí), 1900 (no), 2000 (sí).

## Código en ejecución:



```
Punto1_AñoBisiesto.java [-/A] x
Source History
5 package tp2.programacion.estructurada;
6
7 /**
8  *
9  * @author Usuario
10  */
11 import java.util.Scanner;
12
13 public class Punto1_AñoBisiesto {
14
15     // Método que determina si un año es bisiesto según la regla:
16     // divisible por 4, no por 100 salvo que sea divisible por 400.
17     public static boolean esBisiesto(int anio) {
18         if (anio % 400 == 0) return true;
19         if (anio % 100 == 0) return false;
20         return anio % 4 == 0;
21     }
22
23     public static void main(String[] args) {
24         Scanner sc = new Scanner(System.in);
25         System.out.print("Ingrese un año: ");
26         int anio = sc.nextInt();
27
28         if (esBisiesto(anio)) {
29             System.out.println("El año " + anio + " es bisiesto.");
30         } else {
31             System.out.println("El año " + anio + " no es bisiesto.");
32         }
33
34         sc.close();
35     }
36 }
37
38
Output - TP2-PROGRAMACION-ESTRUCTURADA (run) x
run:
Ingrese un año: 2016
El año 2016 es bisiesto.
BUILD SUCCESSFUL (total time: 54 seconds)
```

## 2. Determinar el Mayor de Tres Números.

**Objetivo:** Comparar tres valores enteros ingresados por el usuario y determinar cuál es el mayor.

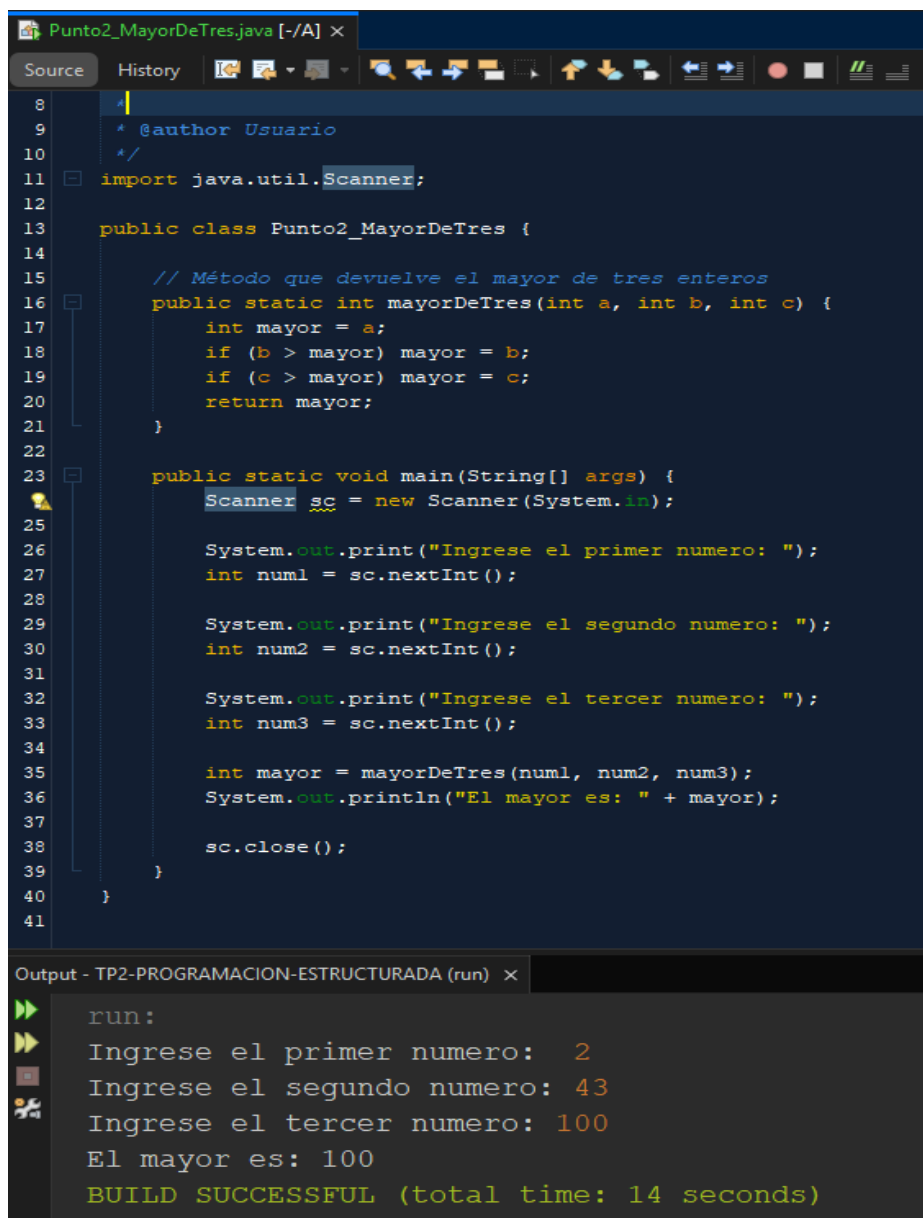
**Algoritmo:** Se inicializa la variable mayor con el primer número, se compara con el segundo y el tercero actualizando si se encuentra un valor superior.

**Entrada:** Tres enteros (num1, num2, num3).

**Salida:** Número mayor encontrado.

**Notas:** Se implementa en un método independiente (mayorDeTres) para modularidad.

### Código en ejecución:



```
Punto2_MayorDeTres.java [-/A] x
Source History
8
9  * @author Usuario
10  */
11  import java.util.Scanner;
12
13  public class Punto2_MayorDeTres {
14
15      // Método que devuelve el mayor de tres enteros
16      public static int mayorDeTres(int a, int b, int c) {
17          int mayor = a;
18          if (b > mayor) mayor = b;
19          if (c > mayor) mayor = c;
20          return mayor;
21      }
22
23      public static void main(String[] args) {
24          Scanner sc = new Scanner(System.in);
25
26          System.out.print("Ingrese el primer numero: ");
27          int num1 = sc.nextInt();
28
29          System.out.print("Ingrese el segundo numero: ");
30          int num2 = sc.nextInt();
31
32          System.out.print("Ingrese el tercer numero: ");
33          int num3 = sc.nextInt();
34
35          int mayor = mayorDeTres(num1, num2, num3);
36          System.out.println("El mayor es: " + mayor);
37
38          sc.close();
39      }
40  }
41
Output - TP2-PROGRAMACION-ESTRUCTURADA (run) x
run:
Ingrese el primer numero: 2
Ingrese el segundo numero: 43
Ingrese el tercer numero: 100
El mayor es: 100
BUILD SUCCESSFUL (total time: 14 seconds)
```

### 3. Clasificación de Edad.

**Objetivo:** Determinar la etapa de vida según la edad ingresada.

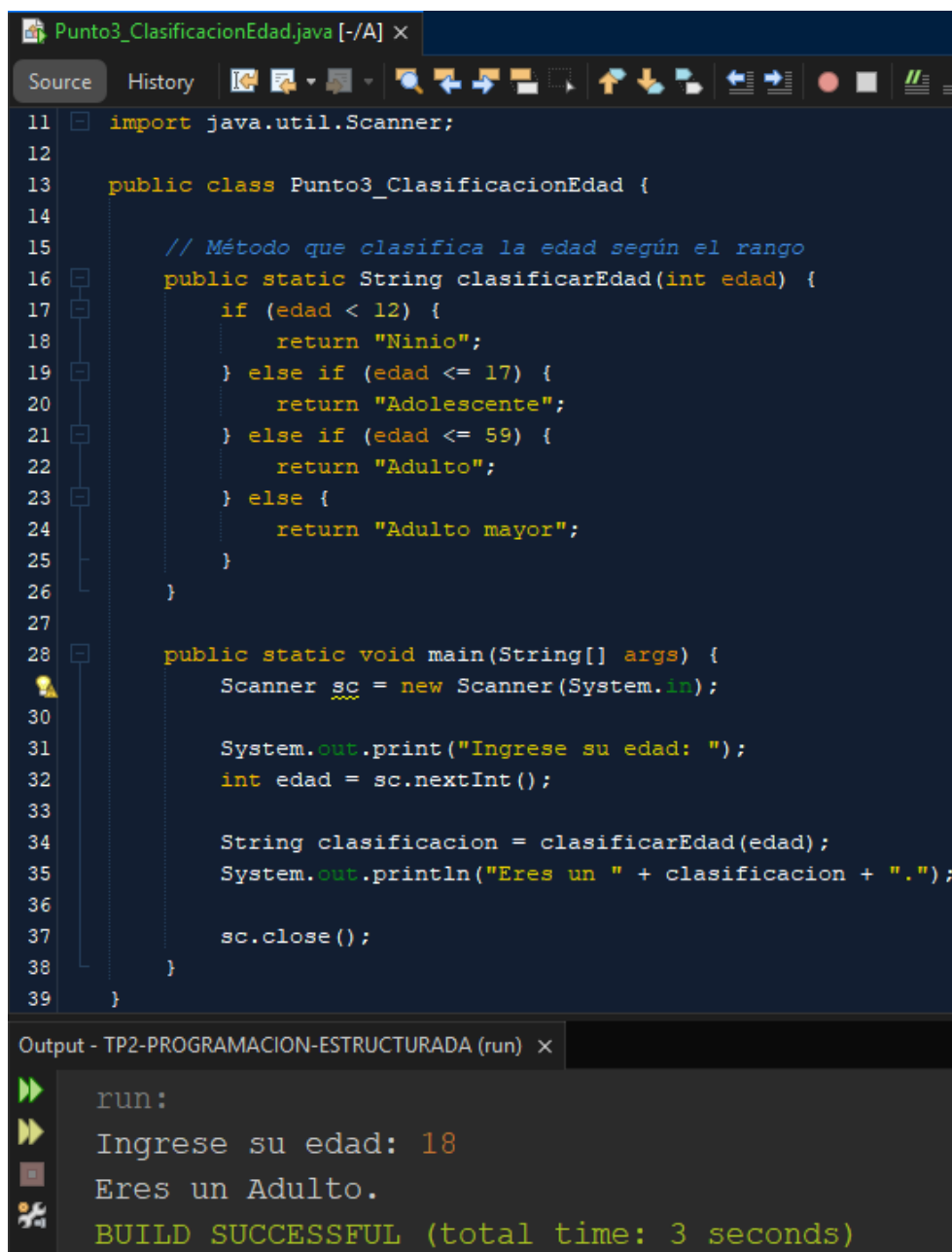
**Algoritmo:** Condicionales if-else que comparan la edad con los rangos predefinidos. Los límites son inclusivos en el extremo superior para no omitir edades exactas como 17 o 59.

**Entrada:** Un entero (edad).

**Salida:** Mensaje con la categoría correspondiente ("Niño", "Adolescente", "Adulto", "Adulto mayor").

**Notas:** Implementado como método separado para facilitar pruebas y reuso.

**Código en ejecución:**



```
Punto3_ClasificacionEdad.java [-/A] x
Source History
11 import java.util.Scanner;
12
13 public class Punto3_ClasificacionEdad {
14
15     // Método que clasifica la edad según el rango
16     public static String clasificarEdad(int edad) {
17         if (edad < 12) {
18             return "Ninio";
19         } else if (edad <= 17) {
20             return "Adolescente";
21         } else if (edad <= 59) {
22             return "Adulto";
23         } else {
24             return "Adulto mayor";
25         }
26     }
27
28     public static void main(String[] args) {
29         Scanner sc = new Scanner(System.in);
30
31         System.out.print("Ingrese su edad: ");
32         int edad = sc.nextInt();
33
34         String clasificacion = clasificarEdad(edad);
35         System.out.println("Eres un " + clasificacion + ".");
36
37         sc.close();
38     }
39 }
```

Output - TP2-PROGRAMACION-ESTRUCTURADA (run) x

```
run:
Ingrese su edad: 18
Eres un Adulto.
BUILD SUCCESSFUL (total time: 3 seconds)
```

## 4. Calculadora de Descuento según categoría.

**Objetivo:** Calcular el precio final de un producto aplicando un descuento según la categoría (A, B o C).

### Algoritmo:

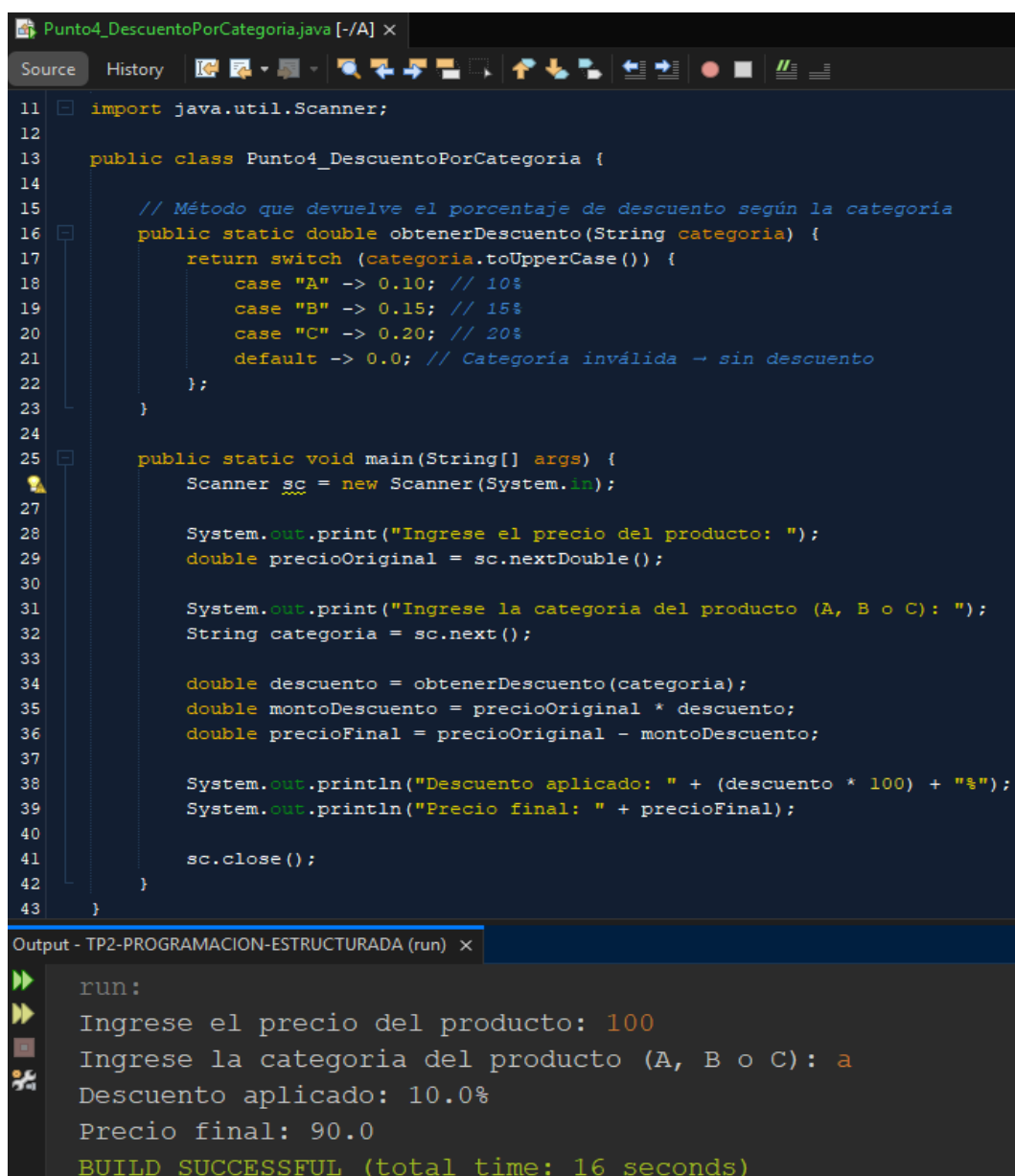
1. Solicitar precio y categoría.
2. Determinar porcentaje de descuento con switch-case.
3. Calcular monto de descuento y restarlo al precio original.

**Entrada:** Precio (double), Categoría (String).

**Salida:** Descuento aplicado y precio final.

**Notas:** Se usa `toUpperCase()` para aceptar categorías ingresadas en minúsculas.

### Código en ejecución:



```
Punto4_DescuentoPorCategoria.java [-/A] x
Source History
11 import java.util.Scanner;
12
13 public class Punto4_DescuentoPorCategoria {
14
15     // Método que devuelve el porcentaje de descuento según la categoría
16     public static double obtenerDescuento(String categoria) {
17         return switch (categoria.toUpperCase()) {
18             case "A" -> 0.10; // 10%
19             case "B" -> 0.15; // 15%
20             case "C" -> 0.20; // 20%
21             default -> 0.0; // Categoría inválida -> sin descuento
22         };
23     }
24
25     public static void main(String[] args) {
26         Scanner sc = new Scanner(System.in);
27
28         System.out.print("Ingrese el precio del producto: ");
29         double precioOriginal = sc.nextDouble();
30
31         System.out.print("Ingrese la categoría del producto (A, B o C): ");
32         String categoria = sc.next();
33
34         double descuento = obtenerDescuento(categoria);
35         double montoDescuento = precioOriginal * descuento;
36         double precioFinal = precioOriginal - montoDescuento;
37
38         System.out.println("Descuento aplicado: " + (descuento * 100) + "%");
39         System.out.println("Precio final: " + precioFinal);
40
41         sc.close();
42     }
43 }
```

Output - TP2-PROGRAMACION-ESTRUCTURADA (run) x

```
run:
Ingrese el precio del producto: 100
Ingrese la categoría del producto (A, B o C): a
Descuento aplicado: 10.0%
Precio final: 90.0
BUILD SUCCESSFUL (total time: 16 seconds)
```

## 5. Suma de Números Pares (while).

**Objetivo:** Sumar únicamente los números pares que el usuario ingrese hasta que se introduzca un cero.

### Algoritmo:

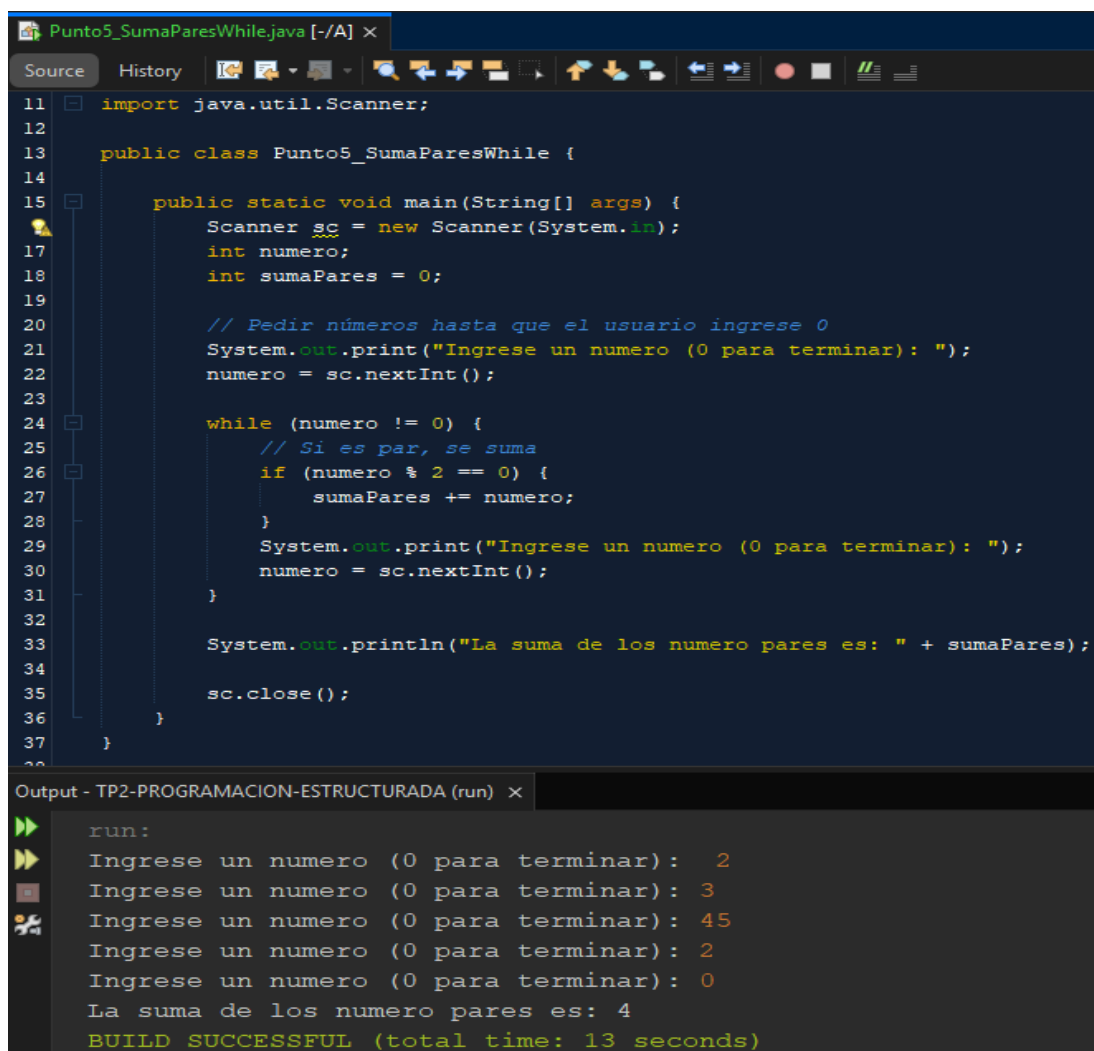
1. Inicializar suma en 0.
2. Leer número.
3. Mientras sea distinto de 0:
  - a. Si es par, agregar a la suma.
  - b. Pedir otro número.
4. Mostrar el total acumulado.

**Entrada:** Números enteros.

**Salida:** Suma de todos los pares ingresados.

**Notas:** Uso de ciclo while porque la cantidad de entradas no se conoce de antemano.

### Código en ejecución:



```
11 import java.util.Scanner;
12
13 public class Punto5_SumaParesWhile {
14
15     public static void main(String[] args) {
16         Scanner sc = new Scanner(System.in);
17         int numero;
18         int sumaPares = 0;
19
20         // Pedir números hasta que el usuario ingrese 0
21         System.out.print("Ingrese un numero (0 para terminar): ");
22         numero = sc.nextInt();
23
24         while (numero != 0) {
25             // Si es par, se suma
26             if (numero % 2 == 0) {
27                 sumaPares += numero;
28             }
29             System.out.print("Ingrese un numero (0 para terminar): ");
30             numero = sc.nextInt();
31         }
32
33         System.out.println("La suma de los numero pares es: " + sumaPares);
34
35         sc.close();
36     }
37 }
```

Output - TP2-PROGRAMACION-ESTRUCTURADA (run) x

```
run:
Ingrese un numero (0 para terminar): 2
Ingrese un numero (0 para terminar): 3
Ingrese un numero (0 para terminar): 45
Ingrese un numero (0 para terminar): 2
Ingrese un numero (0 para terminar): 0
La suma de los numero pares es: 4
BUILD SUCCESSFUL (total time: 13 seconds)
```

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

**Objetivo:** Contar cuántos números positivos, negativos y ceros hay en un conjunto de 10 números ingresados por el usuario.

### Algoritmo:

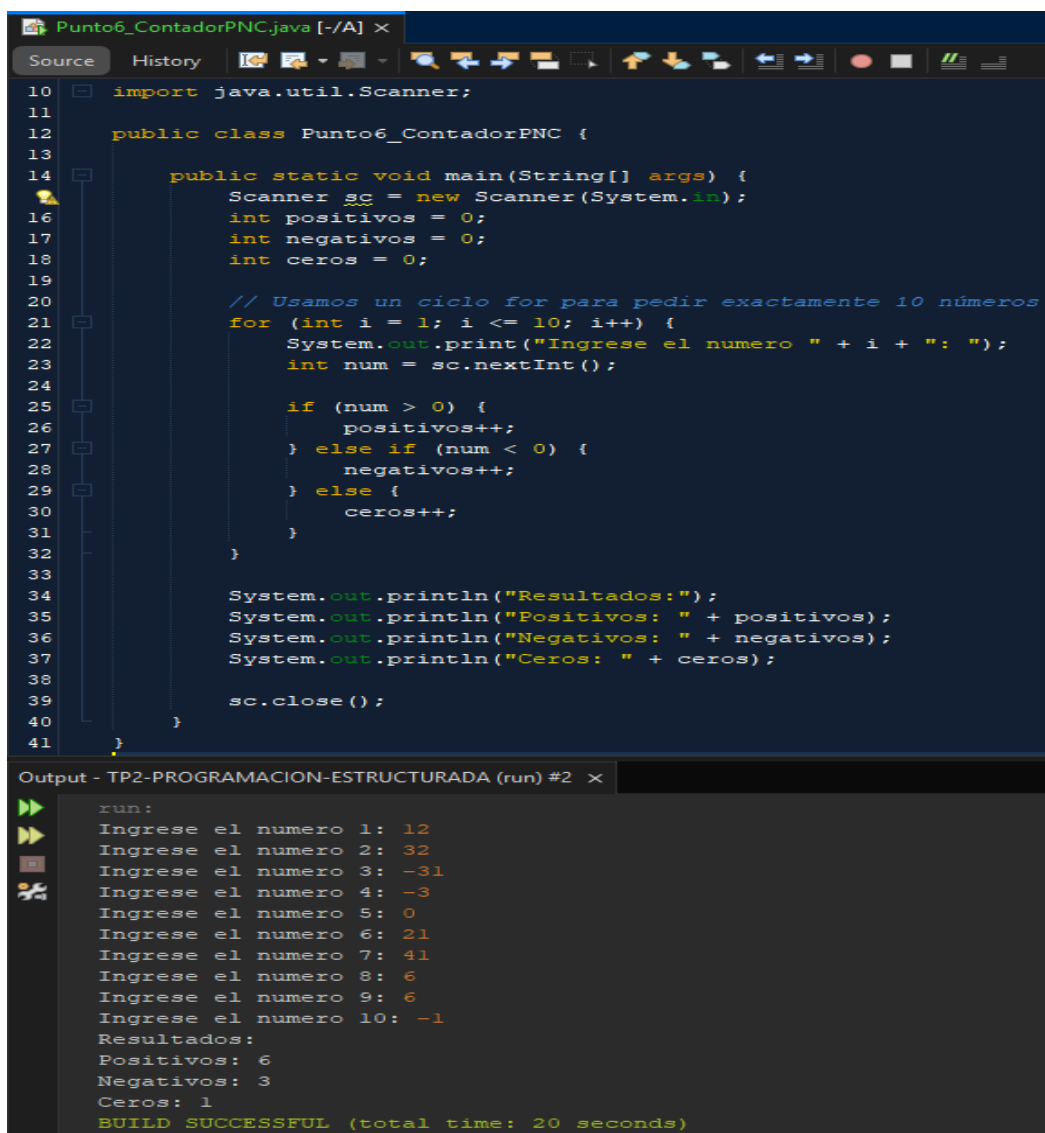
1. Inicializar contadores en 0.
2. Repetir 10 veces:
  - Leer número.
  - Incrementar contador según su signo.
3. Mostrar resultados.

**Entrada:** 10 números enteros.

**Salida:** Cantidad de positivos, negativos y ceros.

**Notas:** Uso de ciclo for porque se conoce la cantidad exacta de iteraciones.

### Código en ejecución:



```
10 import java.util.Scanner;
11
12 public class Punto6_ContadorPNC {
13
14     public static void main(String[] args) {
15         Scanner sc = new Scanner(System.in);
16         int positivos = 0;
17         int negativos = 0;
18         int ceros = 0;
19
20         // Usamos un ciclo for para pedir exactamente 10 números
21         for (int i = 1; i <= 10; i++) {
22             System.out.print("Ingrese el numero " + i + ": ");
23             int num = sc.nextInt();
24
25             if (num > 0) {
26                 positivos++;
27             } else if (num < 0) {
28                 negativos++;
29             } else {
30                 ceros++;
31             }
32         }
33
34         System.out.println("Resultados:");
35         System.out.println("Positivos: " + positivos);
36         System.out.println("Negativos: " + negativos);
37         System.out.println("Ceros: " + ceros);
38
39         sc.close();
40     }
41 }
```

Output - TP2-PROGRAMACION-ESTRUCTURADA (run) #2 x

```
run:
Ingrese el numero 1: 12
Ingrese el numero 2: 32
Ingrese el numero 3: -31
Ingrese el numero 4: -3
Ingrese el numero 5: 0
Ingrese el numero 6: 21
Ingrese el numero 7: 41
Ingrese el numero 8: 6
Ingrese el numero 9: 6
Ingrese el numero 10: -1
Resultados:
Positivos: 6
Negativos: 3
Ceros: 1
BUILD SUCCESSFUL (total time: 20 seconds)
```

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

**Objetivo:** Validar que la nota ingresada esté dentro del rango permitido.

**Algoritmo:**

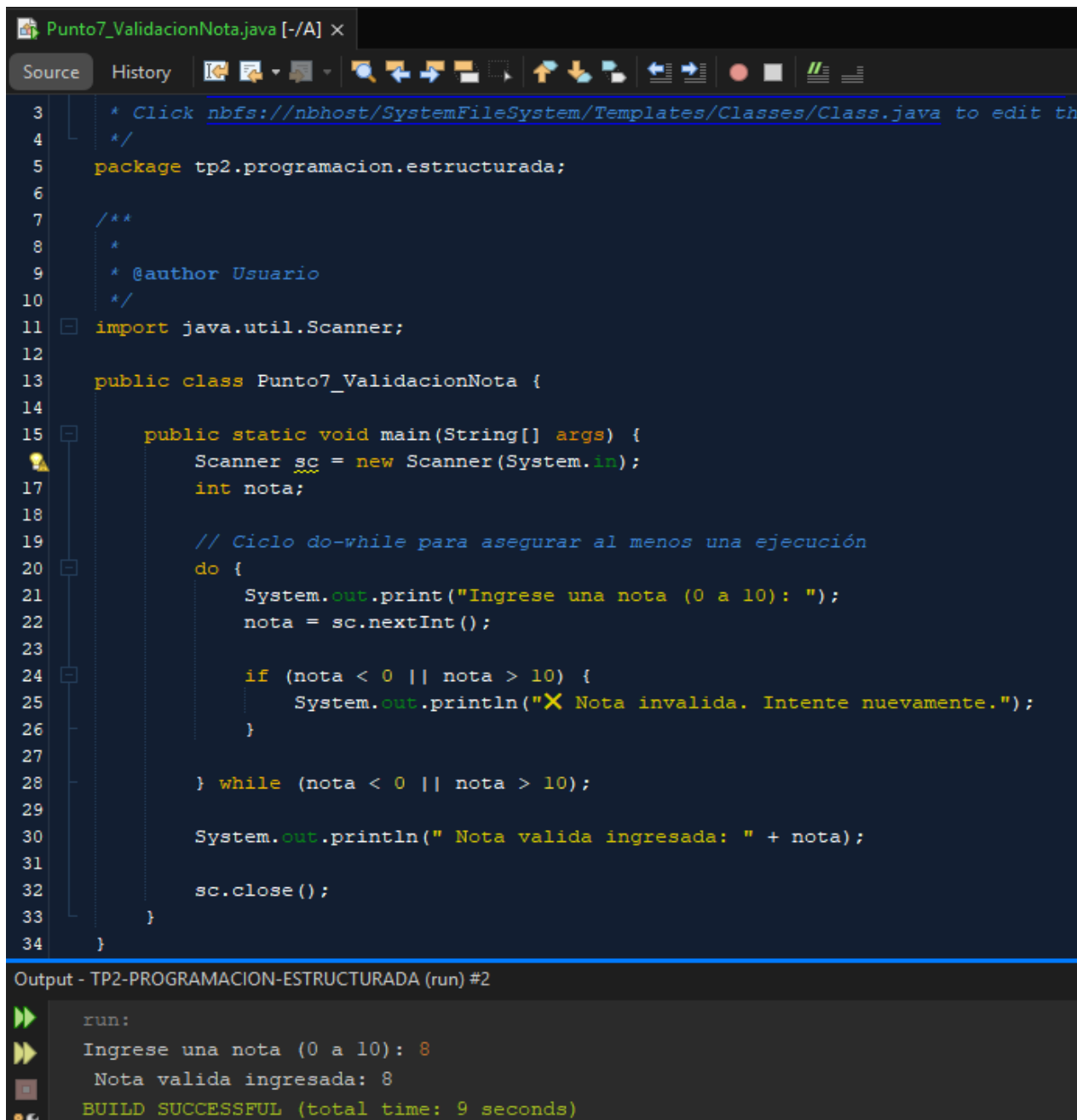
1. Pedir nota.
2. Si no está en el rango 0–10, mostrar error y repetir.
3. Terminar cuando la nota sea válida.

**Entrada:** Un entero (nota).

**Salida:** Confirmación con la nota válida.

**Notas:** Uso de do-while para forzar al menos una lectura antes de validar.

**Código en ejecución:**



```
Punto7_ValidacionNota.java [-/A] x
Source History
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit th
4  */
5  package tp2.programacion.estructurada;
6
7  /**
8   *
9   * @author Usuario
10  */
11  import java.util.Scanner;
12
13  public class Punto7_ValidacionNota {
14
15      public static void main(String[] args) {
16          Scanner sc = new Scanner(System.in);
17          int nota;
18
19          // Ciclo do-while para asegurar al menos una ejecución
20          do {
21              System.out.print("Ingrese una nota (0 a 10): ");
22              nota = sc.nextInt();
23
24              if (nota < 0 || nota > 10) {
25                  System.out.println("X Nota invalida. Intente nuevamente.");
26              }
27
28          } while (nota < 0 || nota > 10);
29
30          System.out.println(" Nota valida ingresada: " + nota);
31
32          sc.close();
33      }
34  }
```

Output - TP2-PROGRAMACION-ESTRUCTURADA (run) #2

```
run:
Ingrese una nota (0 a 10): 8
Nota valida ingresada: 8
BUILD SUCCESSFUL (total time: 9 seconds)
```



## 8. Cálculo del Precio Final con impuesto y descuento.

**Objetivo:** Implementar un método que calcule el precio final a partir del precio base y porcentajes de impuesto y descuento.

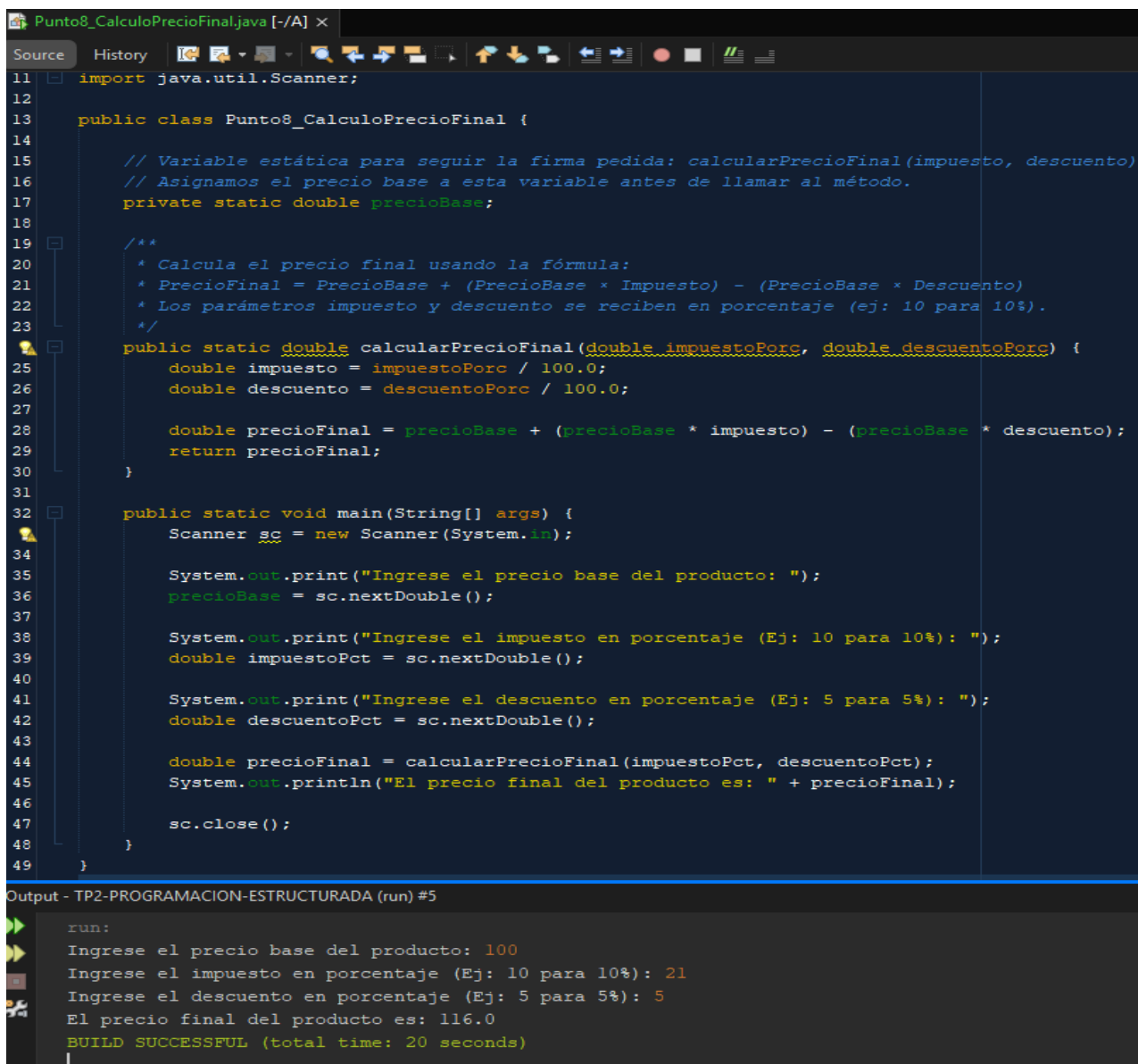
**Algoritmo:** Convertir porcentajes a decimales ( $\div 100$ ) y aplicar la fórmula  $\text{PrecioFinal} = \text{PrecioBase} + (\text{PrecioBase} \times \text{Impuesto}) - (\text{PrecioBase} \times \text{Descuento})$ .

**Entrada:** Precio base (double), impuesto en % (double), descuento en % (double).

**Salida:** Precio final (double).

**Notas:** Se implementa `calcularPrecioFinal(double impuesto, double descuento)` y se mantiene `precioBase` como variable estática para cumplir la firma pedida.

### Código en ejecución:



```
Punto8_CalculoPrecioFinal.java [-/A] x
Source History
11 import java.util.Scanner;
12
13 public class Punto8_CalculoPrecioFinal {
14
15     // Variable estática para seguir la firma pedida: calcularPrecioFinal(impuesto, descuento)
16     // Asignamos el precio base a esta variable antes de llamar al método.
17     private static double precioBase;
18
19     /**
20      * Calcula el precio final usando la fórmula:
21      * PrecioFinal = PrecioBase + (PrecioBase * Impuesto) - (PrecioBase * Descuento)
22      * Los parámetros impuesto y descuento se reciben en porcentaje (ej: 10 para 10%).
23      */
24     public static double calcularPrecioFinal(double impuestoPorc, double descuentoPorc) {
25         double impuesto = impuestoPorc / 100.0;
26         double descuento = descuentoPorc / 100.0;
27
28         double precioFinal = precioBase + (precioBase * impuesto) - (precioBase * descuento);
29         return precioFinal;
30     }
31
32     public static void main(String[] args) {
33         Scanner sc = new Scanner(System.in);
34
35         System.out.print("Ingrese el precio base del producto: ");
36         precioBase = sc.nextDouble();
37
38         System.out.print("Ingrese el impuesto en porcentaje (Ej: 10 para 10%): ");
39         double impuestoPct = sc.nextDouble();
40
41         System.out.print("Ingrese el descuento en porcentaje (Ej: 5 para 5%): ");
42         double descuentoPct = sc.nextDouble();
43
44         double precioFinal = calcularPrecioFinal(impuestoPct, descuentoPct);
45         System.out.println("El precio final del producto es: " + precioFinal);
46
47         sc.close();
48     }
49 }
```

Output - TP2-PROGRAMACION-ESTRUCTURADA (run) #5

```
run:
Ingrese el precio base del producto: 100
Ingrese el impuesto en porcentaje (Ej: 10 para 10%): 21
Ingrese el descuento en porcentaje (Ej: 5 para 5%): 5
El precio final del producto es: 116.0
BUILD SUCCESSFUL (total time: 20 seconds)
```

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

**Objetivo:** Calcular el costo de envío según zona y peso, y luego sumar al precio del producto para obtener el total.

### Funciones implementadas:

calcularCostoEnvio(double peso, String zona) → devuelve el costo de envío aplicando \$5/kg (nacional) o \$10/kg (internacional).

calcularTotalCompra(double precioProducto, double costoEnvio) → devuelve la suma de ambos valores.

**Entrada:** Precio del producto (double), peso (double), zona (String).

**Salida:** Costo de envío (double) y total a pagar (double).

### Código en ejecución:

```
Punto9_CostoEnvioYTotal.java [-:/A] x
Source History
11 import java.util.Scanner;
12
13 public class Punto9_CostoEnvioYTotal {
14
15     // a. Calcula el costo de envío según peso y zona
16     public static double calcularCostoEnvio(double peso, String zona) {
17         double tarifa;
18         if (zona.equalsIgnoreCase("Nacional")) {
19             tarifa = 5.0; // $5 por kg
20         } else if (zona.equalsIgnoreCase("Internacional")) {
21             tarifa = 10.0; // $10 por kg
22         } else {
23             System.out.println("Zona no válida. Usando tarifa nacional por defecto.");
24             tarifa = 5.0;
25         }
26         return peso * tarifa;
27     }
28
29     // b. Suma el precio del producto y el costo de envío
30     public static double calcularTotalCompra(double precioProducto, double costoEnvio) {
31         return precioProducto + costoEnvio;
32     }
33
34     public static void main(String[] args) {
35         Scanner sc = new Scanner(System.in);
36
37         // Solicitar datos
38         System.out.print("Ingrese el precio del producto: ");
39         double precioProducto = sc.nextDouble();
40
41         System.out.print("Ingrese el peso del paquete en kg: ");
42         double pesoPaquete = sc.nextDouble();
43         sc.nextLine(); // limpiar buffer
44
45         System.out.print("Ingrese la zona de envío (Nacional/Internacional): ");
46         String zona = sc.nextLine();
47
48         // Calcular costo de envío
49         double costoEnvio = calcularCostoEnvio(pesoPaquete, zona);
50         System.out.println("El costo de envío es: " + costoEnvio);
51
52         // Calcular total
53         double totalPagar = calcularTotalCompra(precioProducto, costoEnvio);
54     }
55 }
```

Output - TP2-PROGRAMACION-ESTRUCTURADA (run) #5

```
run:
Ingrese el precio del producto: 100
Ingrese el peso del paquete en kg: 5
Ingrese la zona de envío (Nacional/Internacional): Nacional
El costo de envío es: 25.0
El total a pagar es: 125.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 10. Actualización de stock a partir de venta y recepción de productos.

**Objetivo:** Calcular el nuevo stock de un producto después de ventas y recepciones.

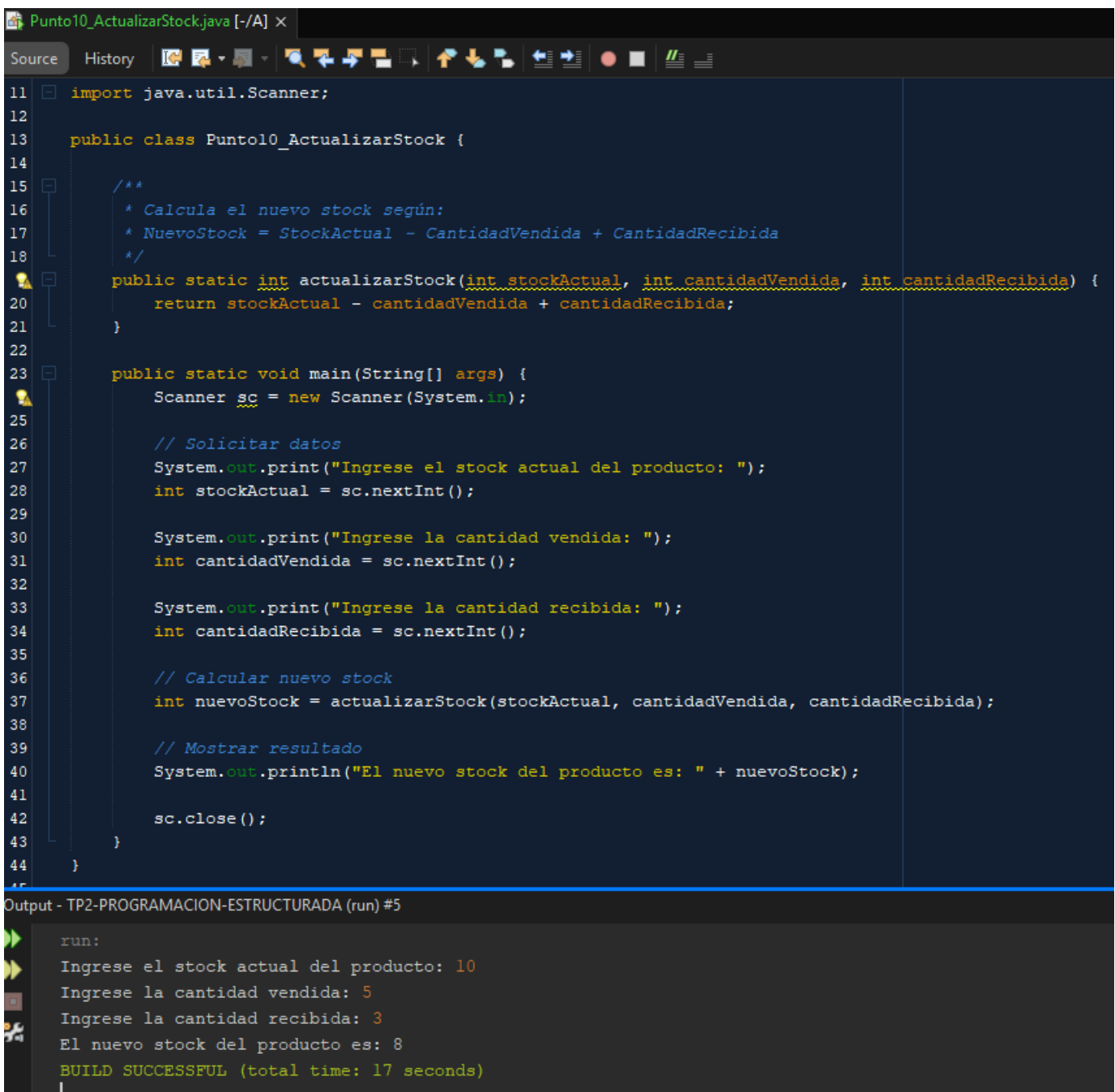
**Fórmula:**  $\text{NuevoStock} = \text{StockActual} - \text{CantidadVendida} + \text{CantidadRecibida}$

**Entrada:** Stock actual (int), cantidad vendida (int), cantidad recibida (int).

**Salida:** Nuevo stock (int).

**Notas:** Implementado en `actualizarStock(int, int, int)` para reutilización y pruebas unitarias.

### Código en ejecución:



```
Punto10_ActualizarStock.java [-/A] x
Source History
11 import java.util.Scanner;
12
13 public class Punto10_ActualizarStock {
14
15     /**
16      * Calcula el nuevo stock según:
17      * NuevoStock = StockActual - CantidadVendida + CantidadRecibida
18      */
19     public static int actualizarStock(int stockActual, int cantidadVendida, int cantidadRecibida) {
20         return stockActual - cantidadVendida + cantidadRecibida;
21     }
22
23     public static void main(String[] args) {
24         Scanner sc = new Scanner(System.in);
25
26         // Solicitar datos
27         System.out.print("Ingrese el stock actual del producto: ");
28         int stockActual = sc.nextInt();
29
30         System.out.print("Ingrese la cantidad vendida: ");
31         int cantidadVendida = sc.nextInt();
32
33         System.out.print("Ingrese la cantidad recibida: ");
34         int cantidadRecibida = sc.nextInt();
35
36         // Calcular nuevo stock
37         int nuevoStock = actualizarStock(stockActual, cantidadVendida, cantidadRecibida);
38
39         // Mostrar resultado
40         System.out.println("El nuevo stock del producto es: " + nuevoStock);
41
42         sc.close();
43     }
44 }

Output - TP2-PROGRAMACION-ESTRUCTURADA (run) #5
run:
Ingrese el stock actual del producto: 10
Ingrese la cantidad vendida: 5
Ingrese la cantidad recibida: 3
El nuevo stock del producto es: 8
BUILD SUCCESSFUL (total time: 17 seconds)
```

## 11. Cálculo de descuento especial usando variable global.

**Objetivo:** Calcular el descuento especial del 10% usando una variable global.

**Variables:**

- Global: DESCUENTO\_ESPECIAL = 0.10
- Local: descuentoAplicado (almacena el valor del descuento calculado).

**Entrada:** Precio del producto (double).

**Salida:** Muestra el descuento aplicado y el precio final (double).

**Código en ejecución:**

```
Punto11_DescuentoEspecial.java [-/A] x
Source History
10  /*
11  import java.util.Scanner;
12
13  public class Puntoll_DescuentoEspecial {
14
15      // Variable global (constante) con el descuento del 10%
16      static final double DESCUENTO_ESPECIAL = 0.10;
17
18      /**
19       * Calcula el precio final aplicando el descuento especial global.
20       * También muestra el descuento aplicado.
21       */
22      public static void calcularDescuentoEspecial(double precio) {
23          double descuentoAplicado = precio * DESCUENTO_ESPECIAL; // variable local
24          double precioFinal = precio - descuentoAplicado;
25
26          System.out.println("El descuento especial aplicado es: " + descuentoAplicado);
27          System.out.println("El precio final con descuento es: " + precioFinal);
28      }
29
30      public static void main(String[] args) {
31          Scanner sc = new Scanner(System.in);
32
33          // Solicitar el precio del producto
34          System.out.print("Ingrese el precio del producto: ");
35          double precio = sc.nextDouble();
36
37          // Calcular y mostrar el descuento
38          calcularDescuentoEspecial(precio);
39
40          sc.close();
41      }
42  }
43
Output - TP2-PROGRAMACION-ESTRUCTURADA (run) #5
run:
Ingrese el precio del producto: 100
El descuento especial aplicado es: 10.0
El precio final con descuento es: 90.0
BUILD SUCCESSFUL (total time: 2 seconds)
```

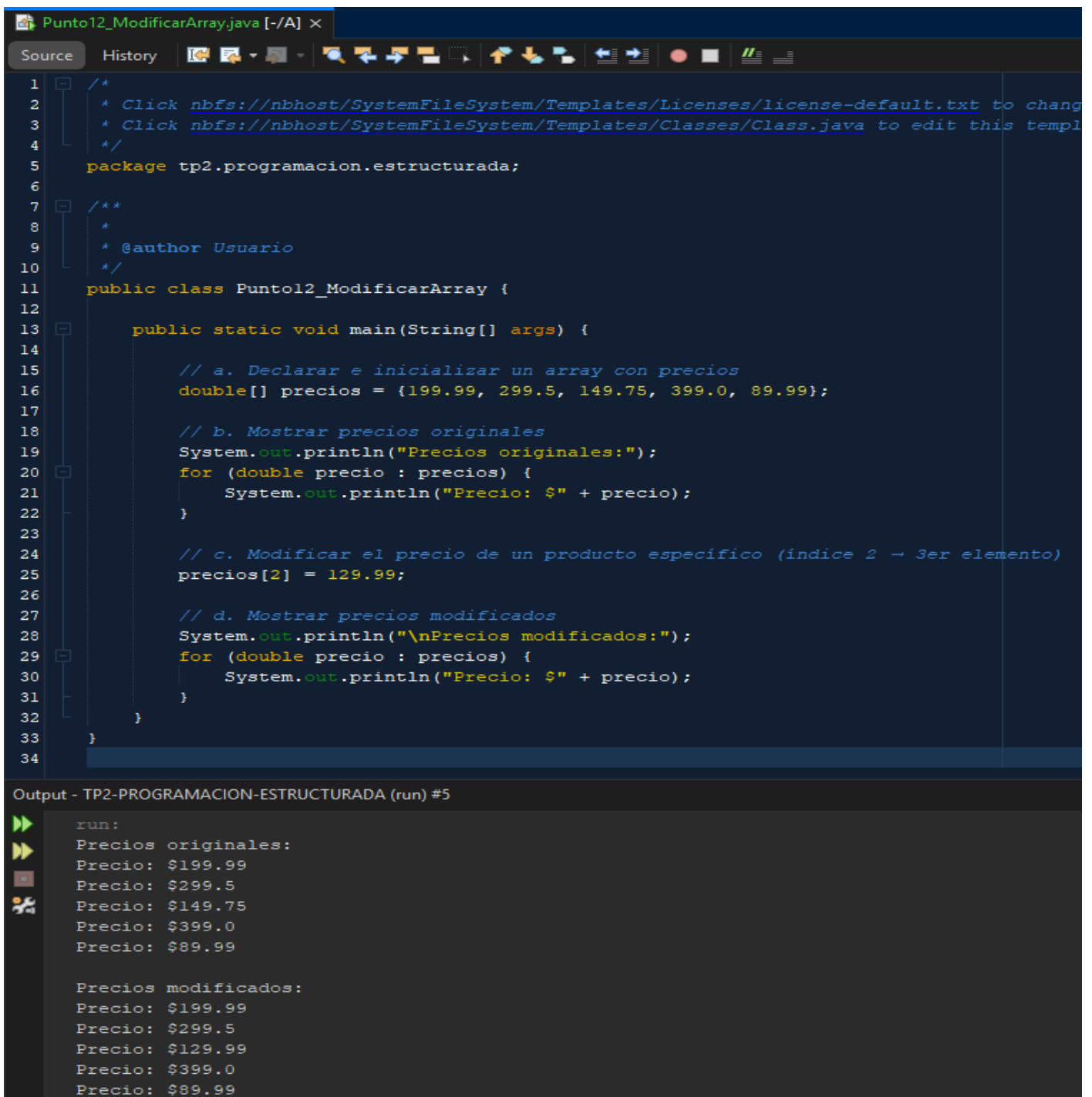
## 12. Modificación de un array de precios y visualización de resultados.

**Objetivo:** Almacenar precios en un array, mostrarlos, modificar un elemento y volver a mostrarlos.

### Conceptos clave:

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

### Código en ejecución:



The screenshot shows an IDE with a Java file named `Punto12_ModificarArray.java`. The code defines a package `tp2.programacion.estructurada` and a class `Punto12_ModificarArray` with a `main` method. The `main` method performs four steps: (a) declares and initializes a `double` array `precios` with values `{199.99, 299.5, 149.75, 399.0, 89.99}`; (b) prints the original prices using a `for-each` loop; (c) modifies the price at index 2 to `129.99`; and (d) prints the modified prices using another `for-each` loop. The output window at the bottom shows the execution results, confirming that the price at index 2 was successfully updated from `$299.5` to `$129.99`.

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5   package tp2.programacion.estructurada;
6
7   /**
8    *
9    * @author Usuario
10   */
11   public class Punto12_ModificarArray {
12
13       public static void main(String[] args) {
14
15           // a. Declarar e inicializar un array con precios
16           double[] precios = {199.99, 299.5, 149.75, 399.0, 89.99};
17
18           // b. Mostrar precios originales
19           System.out.println("Precios originales:");
20           for (double precio : precios) {
21               System.out.println("Precio: $" + precio);
22           }
23
24           // c. Modificar el precio de un producto específico (índice 2 → 3er elemento)
25           precios[2] = 129.99;
26
27           // d. Mostrar precios modificados
28           System.out.println("\nPrecios modificados:");
29           for (double precio : precios) {
30               System.out.println("Precio: $" + precio);
31           }
32       }
33   }
34
```

Output - TP2-PROGRAMACION-ESTRUCTURADA (run) #5

```
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
```

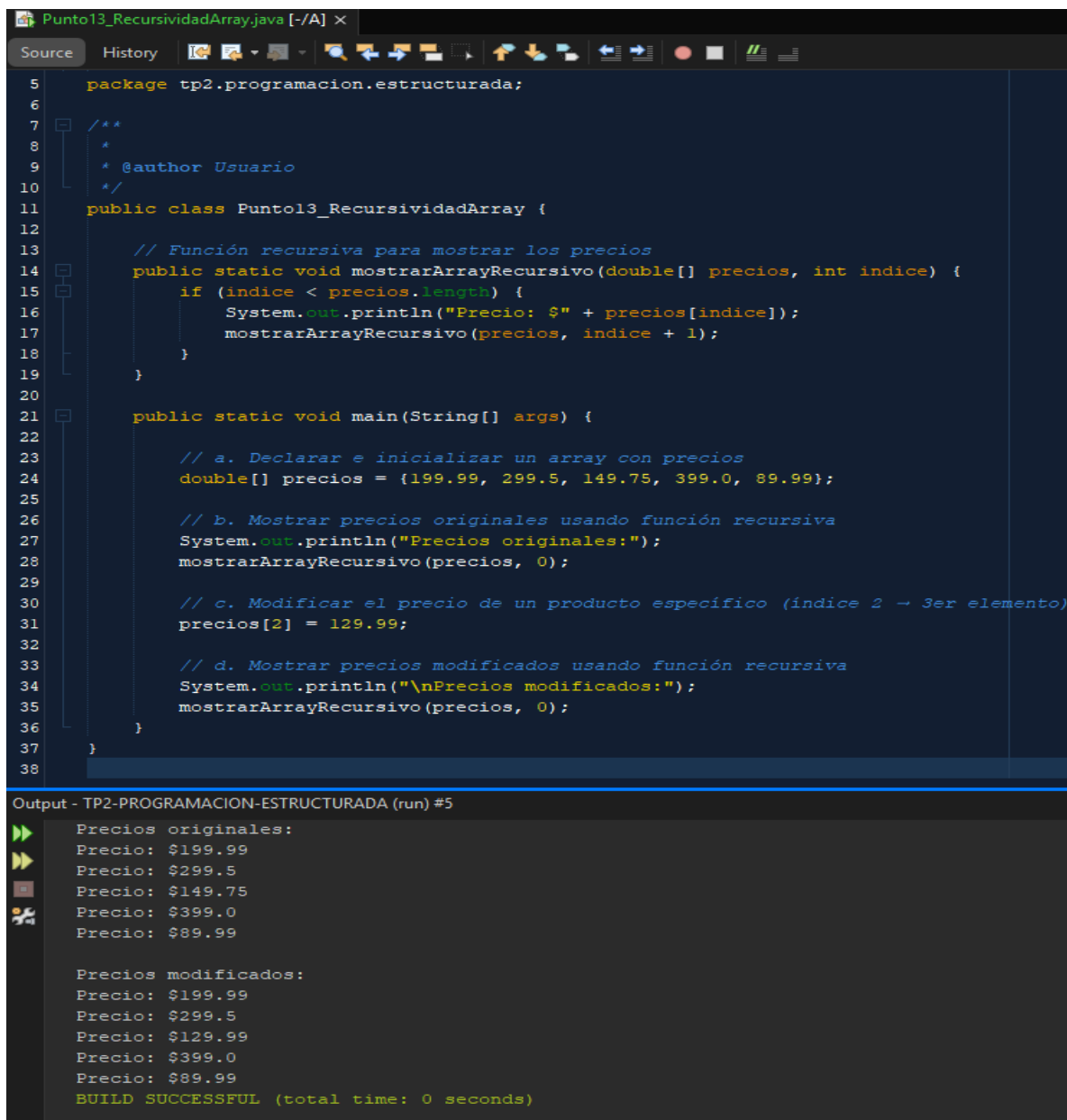
### 13. Impresión recursiva de arrays antes y después de modificar un elemento.

**Objetivo:** Mostrar los elementos de un array antes y después de modificar un valor, utilizando recursividad en lugar de bucles.

#### Conceptos clave:

- ✓ Uso de arrays (double[]) para almacenar valores.
- ✓ Recorrido del array mediante recursión.
- ✓ Modificación de un valor por índice.
- ✓ Uso de un índice como parámetro para controlar la recursión.

#### Código en ejecución:



The screenshot shows an IDE with a Java file named `Punto13_RecursividadArray.java`. The code defines a package `tp2.programacion.estructurada` and a class `Punto13_RecursividadArray`. It includes a recursive method `mostrarArrayRecursivo` that prints array elements and a `main` method that demonstrates its use. The `main` method initializes an array of prices, prints them, modifies the third element (index 2) to 129.99, and prints the array again.

```
5 package tp2.programacion.estructurada;
6
7 /**
8  *
9  * @author Usuario
10 */
11 public class Punto13_RecursividadArray {
12
13     // Función recursiva para mostrar los precios
14     public static void mostrarArrayRecursivo(double[] precios, int indice) {
15         if (indice < precios.length) {
16             System.out.println("Precio: $" + precios[indice]);
17             mostrarArrayRecursivo(precios, indice + 1);
18         }
19     }
20
21     public static void main(String[] args) {
22
23         // a. Declarar e inicializar un array con precios
24         double[] precios = {199.99, 299.5, 149.75, 399.0, 89.99};
25
26         // b. Mostrar precios originales usando función recursiva
27         System.out.println("Precios originales:");
28         mostrarArrayRecursivo(precios, 0);
29
30         // c. Modificar el precio de un producto específico (índice 2 - 3er elemento)
31         precios[2] = 129.99;
32
33         // d. Mostrar precios modificados usando función recursiva
34         System.out.println("\nPrecios modificados:");
35         mostrarArrayRecursivo(precios, 0);
36     }
37 }
38
```

The output of the program is as follows:

```
Output - TP2-PROGRAMACION-ESTRUCTURADA (run) #5
>> 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)
```