



# UNAH

UNIVERSIDAD NACIONAL  
AUTÓNOMA DE HONDURAS



UNIVERSIDAD NACIONAL AUTÓNOMA DE HONDURAS

FACULTAD DE CIENCIAS ECONÓMICAS,  
ADMINISTRATIVAS Y CONTABLES

DEPARTAMENTO DE INFORMÁTICA ADMINISTRATIVA

**ASIGNATURA:**

Programación Intermedia

**CATEDRÁTICO:**

MARVIN JOSUE AGUILAR ROMERO

**ACTIVIDAD:**

Guía de ejercicios de IA

**GRUPO**

Erinson Josué Alvarez Garcia 20211023800

Cristina Lisbeth Cruz Aguilera 20231031044

Kevin Alejandro Laínez Salinas 20231031586

Jonatan Isai Varela Giron 20231000159

Dany Josue Noguera Tinoco 20212020372

**FECHA:**

6/11/2025

## **DISTRIBUCIÓN DE TRABAJO**

EJERCICIO 1: Dany Tinoco

EJERCICIO 2: Kevin Salinas

EJERCICIO 3: Jonatan Varela

EJERCICIO 4: Cristina Cruz

EJERCICIO 5: Erinson Álvarez

EJERCICIO 6: Jonatan Varela

EJERCICIO 7: Dany Tinoco

EJERCICIO 8: Kevin Salinas

EJERCICIO 9: Cristina Cruz

EJERCICIO 10: Erinson Álvarez

EJERCICIO 11: Jonatan Varela

EJERCICIO 12: Dany Tinoco

EJERCICIO 13: Kevin Salinas

EJERCICIO 14: Cristina Cruz

EJERCICIO 15: Erinson Álvarez

EJERCICIO 16: Jonatan Varela

EJERCICIO 17: Dany Tinoco

EJERCICIO 18: Kevin Salinas

EJERCICIO 19: Cristina Cruz

EJERCICIO 20: Erinson Álvarez

EJERCICIO 21: Jonatan Varela

EJERCICIO 22: Dany Tinoco

*EJERCICIO 23: Kevin Salinas*

EJERCICIO 24: Cristina Cruz

EJERCICIO 25: Erinson Álvarez

EJERCICIO 26: Jonatan Varela

EJERCICIO 27: Dany Tinoco

EJERCICIO 28: Kevin Salinas

EJERCICIO 29: Cristina Cruz

EJERCICIO 30: Erinson Álvarez

EJERCICIO 31: Jonatan Varela

EJERCICIO 32: Dany Tinoco

EJERCICIO 33: Kevin Salinas

EJERCICIO 34: Cristina Cruz

EJERCICIO 35: Erinson Álvarez

EJERCICIO 36: Jonatan Varela

EJERCICIO 37: Dany Tinoco

EJERCICIO 38: Kevin Salinas

EJERCICIO 39: Cristina Cruz

EJERCICIO 40: Erinson Álvarez

EJERCICIO 41: Jonatan Varela

EJERCICIO 42: Dany Tinoco

EJERCICIO 43: Kevin Salinas

EJERCICIO 44: Cristina Cruz

EJERCICIO 45: Erinson Álvarez

EJERCICIO 46: Jonatan Varela

EJERCICIO 47: Dany Tinoco

EJERCICIO 48: Kevin Salinas

EJERCICIO 49: Cristina Cruz

EJERCICIO 50: Erinson Álvarez

EJERCICIO 51: Jonatan Varela

EJERCICIO 52: Dany Tinoco

EJERCICIO 53: Kevin Salinas

EJERCICIO 54: Cristina Cruz

EJERCICIO 55: Erinson Álvarez

EJERCICIO 56: Jonatan Varela

EJERCICIO 57: Dany Tinoco

EJERCICIO 58: Kevin Salinas

EJERCICIO 59: Cristina Cruz

EJERCICIO 60: Erinson Álvarez

EJERCICIO 61: Jonatan Varela

EJERCICIO 62: Dany Tinoco

EJERCICIO 63: Kevin Salinas

EJERCICIO 64: Cristina Cruz

EJERCICIO 65: Jonatan Varela

EJERCICIO 66: Dany Tinoco

EJERCICIO 67: Kevin Salinas

EJERCICIO 68: Cristina Cruz

EJERCICIO 69: Erinson Álvarez

EJERCICIO 70: Jonatan Varela

EJERCICIO 71: Dany Tinoco

EJERCICIO 72: Kevin Salinas

EJERCICIO 73: Cristina Cruz

EJERCICIO 74: Jonatan Varela

EJERCICIO 75: Dany Tinoco

EJERCICIO 76: Kevin Salinas

EJERCICIO 77: Cristina Cruz

EJERCICIO 78: Erinson Álvarez

EJERCICIO 79: Jonatan Varela

EJERCICIO 80: Dany Tinoco

EJERCICIO 81: Kevin Salinas

EJERCICIO 82: Cristina Cruz

EJERCICIO 83: Jonatan Varela

EJERCICIO 84: Dany Tinoco

EJERCICIO 85: Kevin Salinas

EJERCICIO 86: Cristina Cruz

EJERCICIO 87: Erinson Álvarez

EJERCICIO 88: Jonatan Varela

EJERCICIO 89: Dany Tinoco

EJERCICIO 90: Kevin Salinas

EJERCICIO 91: Cristina Cruz

EJERCICIO 92: Jonatan Varela

EJERCICIO 93: Dany Tinoco

EJERCICIO 94: Kevin Salinas

EJERCICIO 95: Cristina Cruz

EJERCICIO 96: Jonatan Varela

EJERCICIO 97: Dany Tinoco

EJERCICIO 98: Kevin Salinas

EJERCICIO 99: Cristina Cruz

EJERCICIO 100: Erinson Álvarez

EJERCICIO 101: Jonatan Varela

EJERCICIO 102: Dany Tinoco

EJERCICIO 103: Kevin Salinas

EJERCICIO 104: Cristina Cruz

EJERCICIO 105: Jonatan Varela

EJERCICIO 106: Dany Tinoco

EJERCICIO 107: Kevin Salinas

EJERCICIO 108: Cristina Cruz

EJERCICIO 109: Jonatan Varela

EJERCICIO 110: Dany Tinoco

EJERCICIO 111: Erinson Álvarez

EJERCICIO 112: Jonatan Varela

EJERCICIO 113: Cristina Cruz

EJERCICIO 114: Kevin Salinas

EJERCICIO 115: Jonatan Varela

EJERCICIO 116: Cristina Cruz

EJERCICIO 117: Dany Tinoco

EJERCICIO 118: Kevin Salinas

EJERCICIO 119: Jonatan Varela

EJERCICIO 120: Cristina Cruz

EJERCICIO 121: Dany Tinoco

# DESARROLLO DE EJERCICIOS

## EJERCICIO #1

### Enunciado:

Defina un sello personalizado para el código: Aprovechando la implementación de comentarios en los lenguajes de programación cree un encabezado elegante con el que han de comenzar todos los códigos que usted desarrolle, el mismo debe poseer una versión corta para cuando trabaje modularmente con otras personas. No olvide incluir en el encabezado su nombre, apodo (si gusta), correo electrónico fecha y, en el caso particular de la clase un código correlativo para el ejercicio contenido en ésta guía; además recuerde que el código debe ir documentado por lo que el espacio para análisis/resumen puede incluirse en el encabezado y personalizarse; una vez lo tenga listo usted debe elaborar un programa que muestre un saludo al usuario y luego muestre el encabezado del inciso anterior.

### Análisis de Requerimientos:

- **Sello personalizado:**
  - En Main: Versión completa con nombre, correo (jivarelag@unah.hn), fecha (2025-10-09), código del ejercicio (001), y el enunciado exacto; versión corta con Jonatan Varela | jivarelag@unah.hn | 001 | 2025-10-09.
  - En Utilidades: Solo versión completa y corta, sin el campo PROMPT USADO.
- **Programa:** Mostrar un saludo al usuario y la información del sello personalizado en la consola, siguiendo el formato del ejemplo proporcionado (con secciones como "Metadatos del Desarrollador", "Identificación del Código", "Características del Sello", "Uso del Sello", y la versión corta).
- **Modularidad:** Dividir en dos archivos:
  - Ejercicio001\_Main.java: Flujo principal (saludo y mostrar información del sello).
  - Ejercicio001\_Utilidades.java: Funciones reutilizables para formato (centrar texto y líneas decorativas).
- **Paquete:** Usar package Ejercicio001;
- **Estética:** Bordes Unicode („, „, ||, =) para el saludo, líneas decorativas (=), y texto centrado en un ancho de consola de 70 caracteres, como en el ejemplo.
- **Paradigma:** Programación Estructurada, sin POO, con métodos estáticos y flujo secuencial.
- **Documentación:** Incluir comentarios Javadoc en métodos y el enunciado exacto solo en el sello de Main.

## Constantes:

- ANCHO\_CONSOLA = 70
- LINEA\_HORIZONTAL = '='

## Subrutinas (en Utilidades):

- centrarTexto: Centra texto en la consola.
- mostrarSeparadorDecorado: Imprime una línea decorativa.

**Main:** Secuencia que muestra el saludo, la información del sello, y un mensaje de finalización, siguiendo el formato del ejemplo.

## PROMPT

agente empecemos con el ejercicio #1

ya te cargue mis datos así que ya los tienes, genera una salida muy atractiva, como los ejemplos dados anteriormente.

el enunciado es el siguiente:

Defina un sello personalizado para el código: Aprovechando la implementación de comentarios en los lenguajes de programación cree un encabezado elegante con el que han de comenzar todos los códigos que usted desarrolle, el mismo debe poseer una versión corta para cuando trabaje modularmente con otras personas.

No olvide incluir en el encabezado su nombre, apodo (si gusta), correo electrónico fecha y, en el caso particular de la clase un código correlativo para el ejercicio contenido en ésta guía; además recuerde que el código debe ir documentado por lo que el espacio para análisis/resumen puede incluirse en el encabezado y personalizarse; una vez lo tenga listo usted debe elaborar un programa que muestre un saludo al usuario y luego muestre el encabezado del inciso anterior.

## PROMPT GENERADO

The screenshot shows a code editor interface with two files open:

- Ejercicio001\_Main.java**:

```
/*
 * _____
 * | SELLO PERSONALIZADO - VERSIÓN COMPLETA |
 * | _____
 * | Desarrollador : Jonatan Varela
 * | Correo       : jivarelag@unah.hn
 * | Fecha        : 2025-10-09
 * | Ejercicio    : 001
 * |
 * | PROMPT USADO:
 * | "agente empecemos con el ejercicio #1 ya te cargue mis datos asi que ya los tienes, genera una
 * | salida muy atractiva, como los ejemplos dados anteriormente. el enunciado es el siguiente: Defina un
 * | sello personalizado para el código: Aprovechando la implementación de comentarios en los lenguajes de
 * | programación cree un encabezado elegante con el que han de comenzar todos los códigos que usted
 * | desarrolle, el mismo debe poseer una versión corta para cuando trabaje modularmente con otras
 * | personas. No olvide incluir en el encabezado su nombre, apodo (si gusta), correo electrónico fecha y,
 * | en el caso particular de la clase un código correlativo para el ejercicio contenido en ésta guía;
 * | además recuerde que el código debe ir documentado por lo que el espacio para análisis/resumen puede
 * | incluirse en el encabezado y personalizarse; una vez lo tenga listo usted debe elaborar un programa
 * | que muestre un saludo al usuario and luego muestre el encabezado del inciso anterior."
 * |
 * | _____
 * | SELLO PERSONALIZADO - VERSIÓN CORTA
 * | _____
 * | Jonatan Varela | jivarelag@unah.hn | 001 | 2025-10-09
 * |
 * | _____
```
- Ejercicio001\_Utilidades.java**:

```
/*
 * _____
 * | SELLO PERSONALIZADO - VERSIÓN CORTA
 * | _____
 * | Jonatan Varela | jivarelag@unah.hn | 001 | 2025-10-09
 * |
 * | _____
```

The sidebar on the left shows icons for file operations like Open, Save, and Share, and a search bar. The bottom status bar includes links to Grok help and a file navigation section.

## Salida de la ejecución del programa.

Sello de Jonatan Varela:

```
Guía_de_ejercicios master
Ejercicio001_Main.java

14 * =
15 * /
16 /
17

Identificación del Código:
  • Código Ejercicio : 001

Características del Sello:
  ✓ Incluye versión completa (comentarios al inicio)
  ✓ Incluye versión corta (para trabajo modular)
  ✓ Contiene espacio para análisis/resumen
  ✓ Formato elegante y estandarizado

Usa del Sello:
Este encabezado se utilizará en TODOS los ejercicios desarrollados, asegurando consistencia y profesionalismo en la documentación del código.

Versión Corta (para módulos):
// Jonatan Varela | jivarelag@unah.hn | 001 | 2025-10-09

*** El sello personalizado ha sido definido exitosamente ***

Process finished with exit code 0
```

Sello de Cristina Cruz:

```
Ejercicio001_Main.java
Ejercicio001_Utilidades.java

BIENVENIDO (A) AL SISTEMA DE GESTIÓN DE CÓDIGO!
EJERCICIO 001: MOSTRAR SALUDO Y ENCABEZADOS

ENCABEZADO PERSONALIZADO - VERSIÓN COMPLETA
Autor: Cristina Cruz (Lis <3>)
Correo: cristinacruz@unah.hn
Fecha: 2025-10-10
Código de ejercicio: 001

Análisis: Este programa muestra un saludo y los encabezados personalizados (completo y corto) en cuadros elegantes.

SELLO PERSONALIZADO - VERSIÓN CORTA
Cristina Cruz | Lis <3> | cristinacruz@unah.hn | 001 | 2025-10-10

*** Operaciones completadas exitosamente ***
```

## Sello de Erinson Álvarez

```
Windows PowerShell + - × ne con fines de compatibilidad. Si quieres volver a habilitarlo, ejecuta "Import-Module PSReadLine".  
PS C:\Users\erick> cd "C:\Users\erick\Documents\Proyectos Java\Guía intermedia"  
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> javac -d out src\Ejercicio_001/*.java  
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> java -cp out Ejercicio_001.Main  
  
¡BIENVENIDO AL PROGRAMA!  
Ejercicio 1: Definición de Sello  
  
** ENCABEZADO PERSONALIZADO - VERSIÓN COMPLETA **  
Autor: Erinson Josue Alvarez Garcia (Erinson Alvarez)  
Correo: erinson.alvarez@unah.hn  
Fecha: 2025-10-09  
Código de Ejercicio: 1  
  
** Sello Personalizado - Versión Corta **  
Erinson Alvarez | erinson.alvarez@unah.hn | 001 | 2025-10-09  
Fin de la Demostración del Sello.  
  
--- PROGRAMA FINALIZADO ---  
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## Sello de Kevin Lainez:

```
"C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2.4\lib\idea_rt.jar=65402" -Dfile.encoding=UTF-8  
Bienvenido al Módulo de Programación Estructurada!  
  
♥ INFORMACIÓN DETALLADA DEL SELLO PERSONALIZADO ♥  
  
»» Metadatos del Desarrollador ««  
Desarrollador: Kevin Lainez  
Correo: Klainezs@unah.hn  
  
»» Identificación del Código ««  
Ejercicio No.: 001  
Fecha de Creación: 2025-11-03  
  
»» Características del Sello ««  
Versión Larga: Incluida en cada inicio de archivo (.java)  
Versión Corta: Incluida en la parte superior de los archivos utilitarios  
  
»» Versión Corta del Sello ««  
SELLO PERSONALIZADO - VERSIÓN CORTA  
Kevin Lainez | Klainezs@unah.hn | 001 | 2025-11-03
```

Sello de Dany Noguera:

```
Run - Guia_de_ejercicios
Run Ejercicio001_Main x
Run | Stop | Run | Refresh | Open in new tab | Close

METADATOS DEL DESARROLLADOR
Nombre: Dany Noguera
Correo: dany.noguera@unah.hn
Fecha: 2025-11-04

IDENTIFICACIÓN DEL CÓDIGO
Ejercicio: 001

RESUMEN
Demostración del sello elegante y su versión corta.

VERSIÓN CORTA (para módulos)
Dany Noguera | dany.noguera@unah.hn | 001 | 2025-11-04

¡Se define un sello personalizado!

Process finished with exit code 0
```

## EJERCICIO#2

### Enunciado

Mostrando el sello: El programa anterior no posee salida visible al usuario es por ello que ahora vamos a tomar como base el programa "defina un sello personalizado para el código" y vamos a mostrar al usuario dicho sello.

### Análisis de Requerimientos:

- **Sello personalizado:**
  - En Main: Versión completa con nombre, correo (jivarelag@unah.hn), fecha (2025-10-09), código del ejercicio (002), y el enunciado exacto; versión corta con Jonatan Varela | jivarelag@unah.hn | 002 | 2025-10-09.
  - En Utilidades: Solo versión completa y corta, sin el campo PROMPT USADO.
- **Programa:** Mostrar el sello personalizado (completo y corto) en la consola, basado en el ejercicio #1, con una salida visible y atractiva.
- **Modularidad:** Dividir en dos archivos:
  - Ejercicio002\_Main.java: Flujo principal (mostrar el sello).
  - Ejercicio002\_Utilidades.java: Funciones reutilizables para formato y presentación.
- **Paquete:** Usar package Ejercicio002;
- **Estética:** Bordes Unicode („, „, ||, =) y texto centrado en un ancho de consola de 80 caracteres.
- **Paradigma:** Programación Estructurada, sin POO, con métodos estáticos y flujo secuencial.
- **Documentación:** Incluir comentarios Javadoc en métodos y el enunciado exacto solo en el sello de Main.

### Constantes:

- ANCHO\_CONSOLA = 80
- LINEA\_HORIZONTAL = '='
- BORDE\_VERTICAL = '|'

### Subrutinas (en Utilidades):

- imprimirLineaCentrada: Centra texto en la consola.
- mostrarSeparadorDecorado: Imprime una línea decorativa.
- mostrarSelloElegante: Muestra el sello completo y corto en consola.

**Main:** Secuencia que muestra el título del ejercicio y el sello personalizado.

## PROMPT UTILIZADO

agente ahora el #2 con el siguiente enunciado:

Mostrando el sello: El programa anterior no posee salida visible al usuario es por ello que ahora vamos a tomar como base el programa “defina un sello personalizado para el código” y vamos a mostrar al usuario dicho sello.

muestra un sello elegante como habíamos hecho en el #1, te cuento que tu ya tienes mis datos para ponerlo en el sello, y a ahora muestra el prompt solo en el archivo .Main

## PROMPT GENERADO

### Salida de la ejecución del programa.

Sello de Jonatan Varela:

The screenshot shows an IDE interface with a dark theme. In the top navigation bar, it says "Guia\_de\_ejercicios" and "master". The project structure on the left shows a folder "Guia\_de\_ejercicios" containing ".idea" and "out". The main editor window displays two files: "Ejercicio002\_Main.java" and "Ejercicio002\_Utilidades.java". The code in "Ejercicio002\_Main.java" includes static final variables for line separator and width, and a comment block. The "Run" tab is selected, showing the run configuration "Ejercicio002\_Main". The bottom panel shows the terminal output, which includes a detailed description of the stamp characteristics, instructions for usage, a short version example, and a success message indicating the personalized stamp was shown successfully.

```
private static final String LINEA = "\n"; 2 usages
private static final int ANCHO = 70; 2 usages

/*
 * Características del Sello:
 * ✓ Incluye versión completa (comentarios al inicio)
 * ✓ Incluye versión corta (para trabajo modular)
 * ✓ Contiene espacio para análisis/resumen
 * ✓ Formato elegante y estandarizado
 *
 * Uso del Sello:
 * Este encabezado se utilizará en TODOS los ejercicios
 * desarrollados, asegurando consistencia y profesionalismo
 * en la documentación del código.
 *
 * Version Corta (para módulos):
 *
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 *
 * Autor: Jonatan Varela | jivarelag@unah.hn
 * Fecha: 2025-10-09 | Ejercicio: 002
 *
 * ***
 * El sello personalizado ha sido mostrado exitosamente ***
 */


```

## Sello de Cristina Cruz:

The screenshot shows an IDE interface with two custom seals displayed in separate windows. The top window is titled "SELLO PERSONALIZADO - VERSIÓN CORTA" and contains the text: "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 002 | 2025-10-10". The bottom window is titled "ENCABEZADO PERSONALIZADO - VERSIÓN COMPLETA" and contains detailed metadata: Autor: Cristina Cruz (Lis <3>), Correo: cristinacruz@unah.hn, Fecha: 2025-10-10, Código de ejercicio: 002, and Análisis: Este programa muestra los sellos personalizados (completo y corto) en cuadros elegantes.

## Sello Kevin Lainez:

The screenshot shows an IDE interface displaying a detailed seal for Kevin Lainez. The seal includes the message "¡Iniciando Ejercicio 002: Mostrando el Sello!", followed by a header "INFORMACIÓN DETALLADA DEL SELLO PERSONALIZADO", and sections for developer metadata ("Desarrollador: Kevin Lainez, Correo: klainezs@unah.hn"), code identification ("Ejercicio No.: 002, Fecha de Creación: 2025-11-03"), and short seal version ("SELLO PERSONALIZADO - VERSIÓN CORTA"). The short seal text is "Kevin Lainez | klainezs@unah.hn | 002 | 2025-11-03". The process concludes with a success message: "Ejercicio 002 finalizado con éxito."

## Sello de Dany Noguera:

```
Run - Guia_de_ejercicios
Run Ejercicio002_Main x
G : 
;Tu sello personalizado ahora es visible!
METADATOS DEL DESARROLLADOR
Nombre: Dany Noguera
Correo: dany.noguera@unah.hn
Fecha: 2025-11-04
IDENTIFICACIÓN DEL CÓDIGO
Ejercicio: 002
Mostrando el sello al usuario
ENUNCIADO COMPLETO
El programa anterior no poseía salida visible al usuario.
Ahora mostramos el sello elegante en pantalla.
VERSIÓN CORTA (para módulos)
Dany Noguera | dany.noguera@unah.hn | 002 | 2025-11-04
;SELLO MOSTRADO CON ÉXITO!
```

## Sello de Erinson Alvarez

```
Windows PowerShell x + 
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows
Advertencia: PowerShell detectó que es posible que estés usando un lector de pantalla y que hayas deshabilitado PSReadLine con fines de compatibilidad. Si quieres volver a habilitarlo, ejecuta "Import-Module PSReadLine".
PS C:\Users\erick> cd "C:\Users\erick\Documents\Proyectos Java\Guía intermedia"
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> javac -d out src\Ejercicio_002\*.java
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> java -cp out Ejercicio_002.Main
EJERCICIO 2
MOSTRANDO EL SELLO PERSONALIZADO

** ENCABEZADO PERSONALIZADO - VERSIÓN COMPLETA **

Autor: Erinson Josue Alvarez Garcia
Correo: erinson.alvarez@unah.hn
Fecha: 2025-10-09
Código de Ejercicio: 2

** SELLO PERSONALIZADO - VERSIÓN CORTA **

Erinson Alvarez | erinson.alvarez@unah.hn | 002 | 2025-10-09

<<< PROGRAMA FINALIZADO CON ÉXITO >>>
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> |
```

## EJERCICIO#3

### Enunciado

Aritmética pre-básica: Escriba un programa que pida al usuario que escriba dos números, que obtenga los números del usuario e imprima la suma y la diferencia del primero con el segundo número.

### ANÁLISIS/RESUMEN DEL EJERCICIO

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Jonatan Varela | jivarelag@unah.hn | 003 | 2025-10-09) dentro de los tres archivos; no se incluye **VERSIÓN COMPLETA** en los archivos, pero se muestra en un cuadro elegante durante la ejecución.
- **Programa:** Solicitar dos números al usuario mediante Scanner, calcular suma y diferencia (diferencia en valor absoluto), mostrar en cuatro cuadros separados: bienvenida, ingreso de números, resultados, y sello corto.
- **Modularidad:** Tres archivos en paquete Ejercicio003:
  - Ejercicio003\_Main.java: Flujo principal (entrada de datos y salida).
  - Ejercicio003\_Utilidades.java: Funciones de formato (cuadros para saludo, ingreso, resultados, sello).
  - Ejercicio003\_Operaciones.java: Lógica aritmética (suma y resta).
- **Paquete:** Ejercicio003.
- **Estética:** Ancho de consola de 70 caracteres, bordes Unicode (Γ, Λ, |, -, |) para un diseño simétrico y elegante.
- **Paradigma:** Programación Estructurada, sin POO, con métodos estáticos.
- **Documentación:** Comentarios con sello corto en cada archivo, prompt exacto solo en Ejercicio003\_Main.java.

Constantes:

- ANCHO = 70
- BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '-'

Subrutinas (en Ejercicio003\_Utilidades):

- mostrarSaludo: Muestra un cuadro de bienvenida.
- mostrarIngresoNumeros: Sigue y captura dos números dentro de un cuadro.
- mostrarResultado: Imprime suma y diferencia en un cuadro.
- mostrarSelloCorto: Muestra el sello corto en un cuadro.
- centrarTexto: Centra texto dentro del ancho especificado.
- limpiarLineaConEspacios: Ajusta la línea tras ingresar números para mantenerlos dentro del cuadro.

Subrutinas (en Ejercicio003\_Operaciones):

- sumar: Calcula la suma de dos números.
- restar: Calcula la diferencia en valor absoluto de dos números.

Flujo Principal (en Ejercicio003\_Main):

- Inicia Scanner para entrada de datos.
- Muestra cuadro de bienvenida.
- Muestra cuadro para ingreso de dos números.
- Calcula suma y diferencia con Ejercicio003\_Operaciones.
- Muestra cuadro de resultados.
- Muestra cuadro del sello corto.
- Cierra Scanner y muestra mensaje de finalización.

## PROMPT

"agente el #3 hagamos, te recuerdo nuestro sistema de archivo, en este usa los 3 que son:   |— Ejercicio003\_Main.java   (Programa principal)   |— Ejercicio003\_Utilidades.java   (Funciones de utilidad)   |— Ejercicio003\_Operaciones.java (Lógica del ejercicio) Incluye también nuestro sello en la versión corta hecha en el #2. el enunciado es el siguiente: Aritmética pre-básica: Escriba un programa que pida al usuario que escriba dos números, que obtenga los números del usuario e imprima la suma y la diferencia del primero con el segundo número. que el resultado salga en un cuadro elegante como el sello"

## PROMPT GENERADO

The screenshot shows an AI development interface. On the left, a sidebar titled "Código Generado" lists three files: "Ejercicio003\_Main.java", "Ejercicio003\_Utilidades.java", and "Ejercicio003\_Operaciones.java". A message bubble from the AI says: "NOOO esa solo iba en el #2 ! la VERSIÓN COMPLETA se most... Razonamiento más profundo durante la ejecución". At the bottom, a message says "How can Grok help?". On the right, the "Ejercicio003\_Utilidades.java" code is displayed:

```
package Ejercicio003;

public class Ejercicio003_Utilidades {
    private static final String LINEA = "-";
    private static final int ANCHO = 70;
    private static final char BORDE_IZQ = '|';
    private static final char BORDE_DER = '|';
    private static final char LINEA_H = '=';

    public static void mostrarSaludo() {
        System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
        System.out.println(BORDE_IZQ + centrarTexto("BIENVENIDA AL SISTEMA DE GESTIÓN DE CÓDIGO!", ANCHO - 2) + BORDE_DER);
        System.out.println(BORDE_IZQ + centrarTexto("EJERCICIO 003: ARITMÉTICA PRE-BÁSICA", ANCHO - 2) + BORDE_DER);
        System.out.println("=" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "=");

        public static void mostrarSelloCompleto() {
            System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
            System.out.println(BORDE_IZQ + centrarTexto("SELLO PERSONALIZADO - VERSIÓN COMPLETA", ANCHO - 2) + BORDE_DER);
            System.out.println(BORDE_IZQ + centrarTexto("Desarrollador: Jonatan Varela", ANCHO - 2) + BORDE_DER);
            System.out.println(BORDE_IZQ + centrarTexto("Correo: jivarelag@unah.hn", ANCHO - 2) + BORDE_DER);
            System.out.println(BORDE_IZQ + centrarTexto("Fecha: 2025-10-09", ANCHO - 2) + BORDE_DER);
            System.out.println(BORDE_IZQ + centrarTexto("Ejercicio: 003", ANCHO - 2) + BORDE_DER);
        }
    }
}
```

## Salida de la ejecución del programa:

The screenshot shows the IntelliJ IDEA terminal window displaying the execution output of the generated Java program. The output is as follows:

```
C:\Users\jonat\.jdks\openjdk-24.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2\lib\idea_rt.jar=64222" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8
;BIENVENIDA AL SISTEMA DE GESTIÓN DE CÓDIGO!
EJERCICIO 003: ARITMÉTICA PRE-BÁSICA

INGRESO DE NÚMEROS
Ingrese el primer número: 23
Ingrese el primer número: 23.0
Ingrese el segundo número: 32
Ingrese el segundo número: 32.0

RESULTADOS DE LAS OPERACIONES
Suma: 55.0
Diferencia: 9.0

SELLO PERSONALIZADO - VERSIÓN CORTA
Jonatan Varela | jivarelag@unah.hn | 003 | 2025-10-09
```

## EJERCICIO#4

### Enunciado

Aritmética básica: Escriba un programa que pida al usuario que escriba dos números, que obtenga los números del usuario e imprima la suma, producto, diferencia y cociente de los números.

### Análisis

El código, desarrollado en Java dentro del paquete Ejercicio004, consta de tres clases: Ejercicio004\_Main, Ejercicio004\_Utilidades y Ejercicio004\_Operaciones. Su propósito es realizar operaciones aritméticas básicas (suma, producto, diferencia y cociente) con dos números ingresados por el usuario, mostrando un sello personalizado con información del autor y resultados formateados. La clase Ejercicio004\_Main coordina la ejecución, invocando métodos de las otras clases. Ejercicio004\_Utilidades maneja la entrada de datos mediante Scanner con validación de números válidos, muestra un sello personalizado con texto centrado y líneas divisorias, y presenta los resultados con formato de dos decimales, verificando la división por cero. Ejercicio004\_Operaciones encapsula las operaciones aritméticas en métodos estáticos, retornando Double.NaN para divisiones por cero. El código destaca por su modularidad, con responsabilidades claras por clase, robustez en el manejo de errores (entradas inválidas y división por cero), y una presentación profesional con texto alineado. Es un ejemplo educativo de programación orientada a objetos, ideal para demostrar entrada/salida, manejo de excepciones y organización de código en Java.

### PROMPT

Crea un programa en Java dentro del paquete Ejercicio004, compuesto por tres clases: Ejercicio004\_Main, Ejercicio004\_Utilidades y Ejercicio004\_Operaciones. El programa debe realizar operaciones aritméticas básicas (suma, producto, diferencia y cociente) con dos números de tipo double ingresados por el usuario. Ejercicio004\_Main debe contener el método main para coordinar la ejecución, solicitando los números, invocando las operaciones y mostrando los resultados. Ejercicio004\_Utilidades debe incluir métodos estáticos para leer números válidos con manejo de excepciones, mostrar el sello personalizado (versión corta) antes realizado, y presentar los resultados con formato de dos decimales, verificando la validez del cociente. Ejercicio004\_Operaciones debe tener métodos estáticos para calcular las operaciones, retornando Double.NaN para división por cero. El programa debe ser modular, manejar errores en entradas y división por cero,

mostrar una salida formateada con sello y resultados alineados, y seguir buenas prácticas de programación orientada a objetos

## PROMPT GENERADO

The screenshot shows a Java code editor interface with the following sections:

- Compile all files:**

```
bash x D
javac Ejercicio004/*.j
```
- Run:**
  - From the same directory, run the program:

```
bash x D
java Ejercicio004.Ejer
```
- Expected Output:**

```
text x D
-----| SELLITO PERSONALIZADO
```
- Code Area:**

```
/*
 * Cristina Cruz | Lis | cristinacruz@unah.hn | 004 | 2025-10-15
 */
package Ejercicio004;

public class Ejercicio004_Main {
    public static void main(String[] args) {
        // Mostrar el sello personalizado
        Ejercicio004_Utilidades.mostrarSelloCorto("004", "2025-10-15");

        // Entrada de datos
        double numero1 = Ejercicio004_Utilidades.leerNumero("Ingrese el primer número: ");
        double numero2 = Ejercicio004_Utilidades.leerNumero("Ingrese el segundo número: ");

        // Procesamiento
        double suma = Ejercicio004_Operaciones.calcularSuma(numero1, numero2);
        double producto = Ejercicio004_Operaciones.calcularProducto(numero1, numero2);
        double diferencia = Ejercicio004_Operaciones.calcularDiferencia(numero1, numero2);
        double cociente = Ejercicio004_Operaciones.calcularCociente(numero1, numero2);

        // Salida de resultados
        Ejercicio004_Utilidades.mostrarResultados(numero1, numero2, suma, producto,
                                                diferencia, cociente);
    }
}
```

Salida de la ejecución del programa:

Run Ejercicio004\_Main

```
C:\Users\MATA\.jdks\openjdk-25\bin\java.exe "-javaagent:C:\Program Files\Java\VisualVM\lib\visualvm-agent.jar" -Djava.awt.headless=true Ejercicio004_Main
-----
|           SELLO PERSONALIZADO - VERSIÓN CORTA
|
|   Cristina Cruz | Lis | cristinacruz@unah.hn | 004 | 2025-10-10 |
-----
Ingrese el primer número: 10
Ingrese el segundo número: 2

Resultados de las operaciones:
Suma de 10.00 y 2.00: 12.00
Producto de 10.00 y 2.00: 20.00
Diferencia de 10.00 y 2.00: 8.00
Cociente de 10.00 y 2.00: 5.00

Process finished with exit code 0
```

## EJERCICIO #5

### **enunciado:**

Buscando al mayor: Escriba un programa que pida al usuario que escriba dos enteros, que obtenga los números del usuario e imprima el número más grande, seguido de las palabras "es más grande". Si los números son iguales, imprima el mensaje "Estos números son iguales." (Ejercicio 2.18 Libro 1)

### **análisis:**

Este programa tiene como objetivo principal solicitar dos números enteros al usuario, compararlos y determinar cuál es el mayor, o indicar si son iguales. El diseño está modularizado en tres clases: Main (maneja el flujo principal e interacción), Operaciones (contiene la lógica de comparación) y Utilidades (gestiona la entrada/salida estandarizada y mensajes). Las entradas son dos enteros (num1, num2). Las salidas son una cadena de texto que indica el número mayor seguido de "es más grande" o el mensaje "Estos números son iguales" si son idénticos. El caso límite se da cuando ambos números son iguales. La complejidad es mínima, involucrando solo estructuras condicionales simples () y gestión básica de I/O.

### **prompt**

Crea un programa en Java que solicite al usuario dos números enteros. El programa debe comparar los números e imprimir un mensaje indicando cuál es el mayor. Si los números son iguales, debe imprimir un mensaje de igualdad. Estructura el código en tres clases: una para la lógica de comparación, una para las utilidades de entrada/salida y la clase principal (Main).

### **prompt generado**

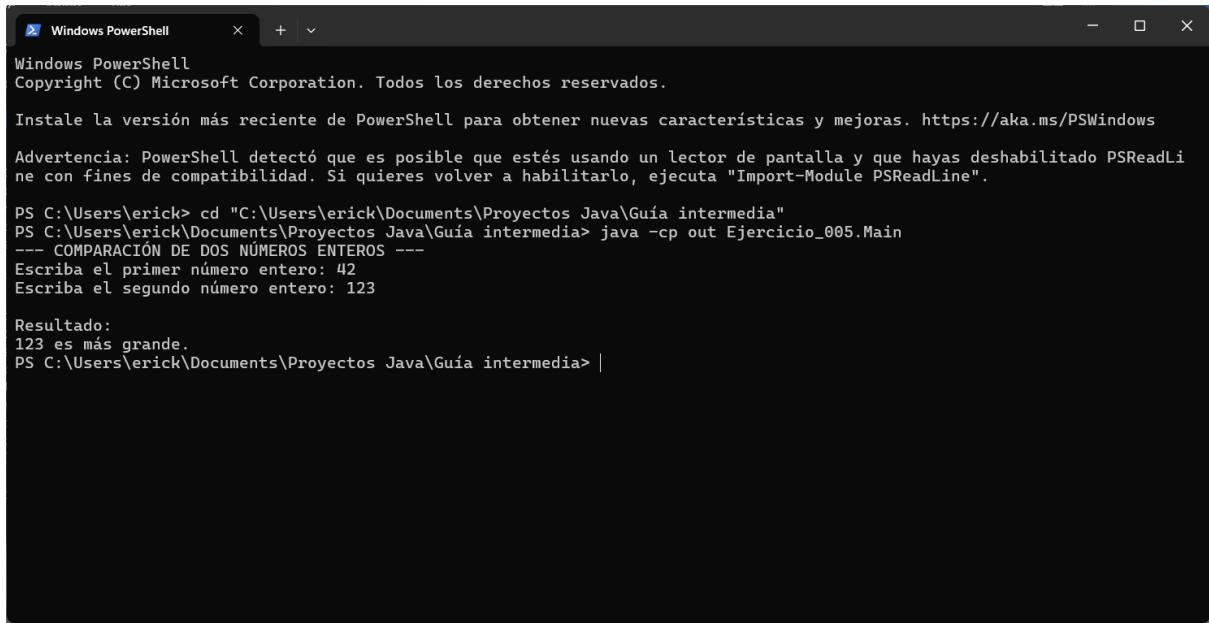
Para cumplir con su solicitud, estructuré la solución modularmente en tres clases distintas.

La clase Utilidades fue diseñada para manejar toda la interacción con el usuario. Proporciona métodos estáticos para mostrar un mensaje de bienvenida, solicitar y capturar de forma limpia un número entero usando un Scanner, y finalmente, un método para mostrar el resultado final de la comparación en un formato claro.

La clase Operaciones encapsula la lógica central. Le añadí un método público que acepta los dos números enteros ingresados. Dentro de este método, usé una estructura de control if-else if-else para realizar las tres posibles comparaciones: Finalmente, la clase Main actúa como el orquestador. Inicializa el Scanner, llama a Utilidades para desplegar el mensaje inicial y para obtener los dos valores de entrada del usuario. Luego, crea una instancia de Operaciones, llama al método de

comparación para obtener la cadena de resultado y, por último, utiliza de nuevo Utilidades para presentar el resultado final en la consola.

Salida:

A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows the following command-line interaction:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

Advertencia: PowerShell detectó que es posible que estés usando un lector de pantalla y que hayas deshabilitado PSReadLine con fines de compatibilidad. Si quieres volver a habilitarlo, ejecuta "Import-Module PSReadLine".

PS C:\Users\erick> cd "C:\Users\erick\Documents\Proyectos Java\Guía intermedia"
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> java -cp out Ejercicio_005.Main
--- COMPARACIÓN DE DOS NÚMEROS ENTEROS ---
Escriba el primer número entero: 42
Escriba el segundo número entero: 123

Resultado:
123 es más grande.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> |
```

The window has a dark theme and standard Windows window controls at the top.

## EJERCICIO#6

### Enunciado

Aritmética pre-básica: Escriba un programa que pida al usuario que escriba dos números, que obtenga los números del usuario e imprima la suma y la diferencia del primero con el segundo número. Mejoras: Implementar "Aritmética trujeada" para restar siempre el menor al mayor, evitar división por cero con mensaje, mostrar números ordenados de mayor a menor, y realizar y mostrar las operaciones aritméticas posibles (suma, resta y división).

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Jonatan Varela | jivarelag@unah.hn | 006 | 2025-10-09) dentro de los tres archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Solicitar dos números al usuario mediante Scanner, ordenarlos de mayor a menor, calcular suma, resta (mayor-menor), y división (solo si el menor no es cero, con mensaje de advertencia si lo es). Mostrar resultados en cuadros elegantes.
- **Modularidad:** Tres archivos en paquete Ejercicio006:
  - Ejercicio006\_Main.java: Flujo principal con entrada de datos, ordenamiento, cálculos y salida; incluye el prompt.
  - Ejercicio006\_Utilidades.java: Funciones de formato (cuadros para saludo, ingreso, resultados, sello).
  - Ejercicio006\_Operaciones.java: Lógica aritmética (suma, resta, división).
- **Paquete:** Ejercicio006.
- **Estética:** Ancho de consola de 70 caracteres, bordes Unicode (Γ, Λ, Τ, Τ, −, |) para un diseño simétrico y elegante.
- **Paradigma:** Programación Estructurada, métodos estáticos.
- **Documentación:** Sello corto en cada archivo, prompt exacto en Ejercicio006\_Main.java.

### Constantes:

- ANCHO = 70
- BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '-'

### Subrutinas (en Ejercicio006\_Utilidades):

- mostrarSaludo: Muestra un cuadro de bienvenida.
- mostrarIngresoNumeros: Solicita y capture dos números dentro de un cuadro (pendiente de ajuste para alineación).
- mostrarResultado: Imprime números ordenados y resultados (suma, resta, división condicional) en cuadros.
- mostrarSelloCorto: Muestra el sello corto en un cuadro.
- centrarTexto: Centra texto dentro del ancho especificado.
- limpiarLineaConEspacios: Ajusta la línea tras ingresar números (a mejorar para alineación).

### **Subrutinas (en Ejercicio006\_Operaciones):**

- sumar: Calcula la suma de dos números.
- restar: Calcula la diferencia de dos números.
- dividir: Realiza la división de dos números.

### **Flujo Principal (en Ejercicio006\_Main):**

- Inicia Scanner para entrada de datos.
- Muestra cuadro de bienvenida.
- Muestra cuadro para ingreso de dos números.
- Determina mayor y menor con Math.max y Math.min.
- Calcula suma, resta (mayor-menor), y división (si el menor no es cero).
- Muestra cuadro con números ordenados y resultados.
- Muestra cuadro del sello corto.
- Cierra Scanner y muestra mensaje de finalización.

## **PROMPT**

Agente necesito que realicemos el ejercicio #6, te recuerdo que en todo deben de ir el sello anteriormente hecho, y nuestra distribución de 3 archivos:

```

└── Ejercicio###/
    ├── Ejercicio###_Main.java      → Control del flujo principal (Acá debe de ir el
                                    prompt)
    ├── Ejercicio###_Utilidades.java → Cuadros, sellos y formato estético.
    └── Ejercicio###_Operaciones.java → Lógica matemática o funcional.

```

Necesito que mejoremos este ejercicio:

Aritmética pre-básica: Escriba un programa que pida al usuario que escriba dos números, que obtenga los números del usuario e imprima la suma y la diferencia del primero con el segundo número.

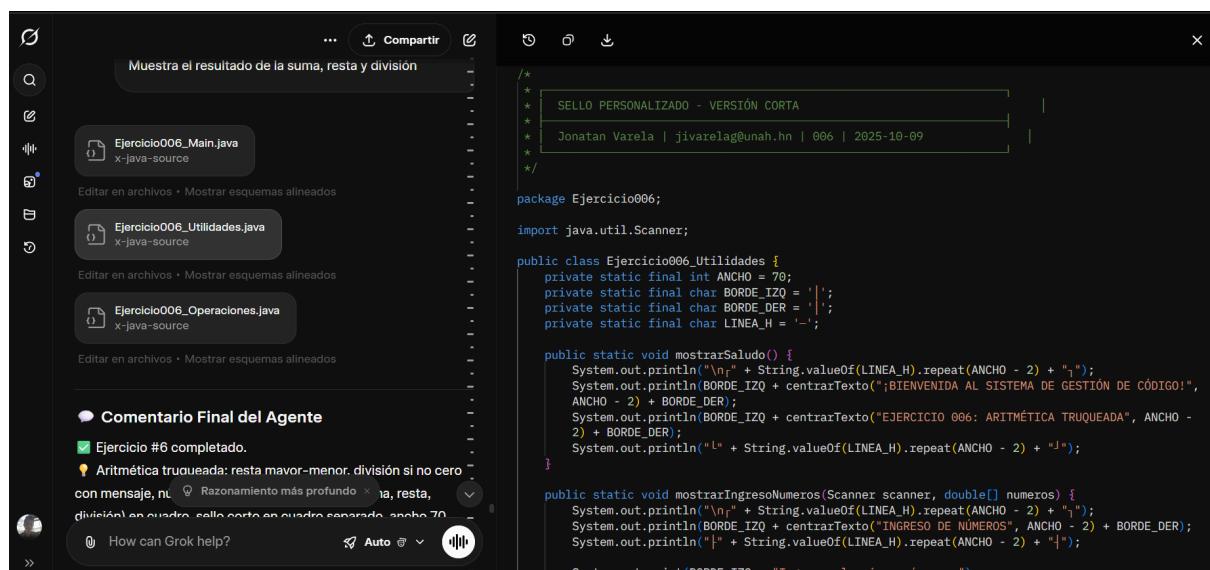
debemos de implementar estas mejoras:

Aritmética truqueada: permitir que de manera que siempre se le reste al mayor el menor y que nunca permita la división por cero, para esto si será posible dividir los números lo hará sin problemas si no se le mostrará un mensaje al usuario que no se mostrará la división luego, mostrará los números ordenados de mayor a menor y procederá a realizar y mostrar las operaciones aritméticas posibles.

También brinda un mensaje que no se puede dividir entre 0, pero que el programa siga corriendo, al hacer la resta siempre se le reste al mayor el menor, junto con todo lo demás que te corresponde hacer.

Muestra el resultado de la suma, resta y división

## PROMPT GENERADO



The screenshot shows a code editor interface with several files listed on the left and a large code block on the right. The files include `Ejercicio006_Main.java`, `Ejercicio006_Utilidades.java`, and `Ejercicio006_Operaciones.java`. The main code block contains Java code for a personalized stamp and an arithmetic exercise. A comment at the top of the code block reads:

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 006 | 2025-10-09
 */
```

The code itself includes imports for `java.util.Scanner`, defines constants for width and border characters, and implements two static methods: `mostrarSaludo()` and `mostrarIngresoNumeros(Scanner scanner, double[] numeros)`.

## Salida de la ejecución del programa:

```
Ingrese el primer número: 43
Ingrese el primer número: 43.0
Ingrese el segundo número: 23
Ingrese el segundo número: 23.0

NÚMEROS ORDENADOS (MAYOR A MENOR)
Mayor: 43.0 | Menor: 23.0

RESULTADOS DE LAS OPERACIONES
Suma: 66.0
Diferencia: 20.0
División: 1.8695652173913044

SELLO PERSONALIZADO - VERSIÓN CORTA
Jonatan Varela | jivarelag@unah.hn | 006 | 2025-10-09

*** Operaciones completadas exitosamente ***
```

## EJERCICIO#7

### Enunciado

Escriba un programa que reciba tres enteros del teclado e imprima la suma, promedio, producto, menor y mayor de esos números.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 007 | 2025-10-09) dentro de los tres archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Solicitar tres números al usuario mediante Scanner, imprima la suma, promedio, producto, menor y mayor de esos números.. Mostrar resultados en cuadros elegantes.
- **Modularidad:** Tres archivos en paquete Ejercicio007:
  - Ejercicio007\_Main.java: Flujo principal con entrada de datos, ordenamiento, cálculos y salida; incluye el prompt.
  - Ejercicio007\_Utilidades.java: Funciones de formato (cuadros para saludo, ingreso, resultados, sello).
  - Ejercicio007\_Operaciones.java: Lógica aritmética (suma, promedio, producto, mayor y menor).
- **Paquete:** Ejercicio007.
- **Estética:** Ancho de consola de 70 caracteres, bordes Unicode (Γ, Λ, Τ, Σ, −, |) para un diseño simétrico y elegante.
- **Paradigma:** Programación Estructurada, métodos estáticos.

- **Documentación:** Sello corto en cada archivo, prompt exacto en Ejercicio007\_Main.java.

Constantes:

- ANCHO = 70
- BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '-'

### **Subrutinas (en Ejercicio007\_Utilidades):**

- Recibe textos para encabezados, separadores o pie.
- Genera bordes superiores o inferiores con caracteres Unicode.
- Centra el texto dentro del ancho definido.
- Imprime en consola.

### **Subrutinas (en Ejercicio007\_Operaciones):**

- Recibe los tres números como parámetros.
- Ejecuta la operación matemática correspondiente:
  - sumar,
  - promediar,
  - multiplicar
  - encontrar el mínimo,
  - encontrar el máximo.
- Retorna el resultado al método llamador (Main.java).
- No imprime nada en consola.

### **Flujo Principal (en Ejercicio007\_Main):**

- Mostrar sello personalizado con borde decorado (mostrarEncabezado()).
- Solicitar al usuario tres valores numéricos.
- Enviar esos valores a los métodos de Operaciones.java:
  - – suma()

- promedio()
- producto()
- menor()
- mayor()
- Mostrar resultados en consola usando funciones visuales de Utilidades.java.
- Mostrar pie de página decorado (mostrarPie()).

## PROMPT

Agente necesito que realicemos el ejercicio #7, te recuerdo que en todo deben de ir el sello anteriormente hecho, y nuestra distribución de 3 archivos:

```

└── Ejercicio###/
    ├── Ejercicio###_Main.java      → Control del flujo principal (Acá debe de ir el
    prompt)
    ├── Ejercicio###_Utilidades.java → Cuadros, sellos y formato estético.
    └── Ejercicio###_Operaciones.java → Lógica matemática o funcional.

```

Ejercicio #7 Escriba un programa que reciba tres enteros del teclado e imprima la suma, promedio, producto, menor y mayor de esos números.

## PROMPT GENERADO

The screenshot shows a code editor interface with the following details:

- Left Sidebar:** Shows tabs for "Instrucciones" (selected), "Archivos" (selected), "Conversaciones", and "Adjuntar". Below these are buttons for "Adjuntar" and "Ejercicios IAO...".
- Central Area:**
  - A message box says: "¡Claro, agente! Aquí tienes el Ejercicio #7 100 % listo para IntelliJ, con la estructura exacta de 3 archivos que acordamos, sello elegante completo, versión corta, y todo funcionando perfecto."
  - A section titled "Ejercicio #7 – Jugando con tres números" contains the instruction: "Recibe 3 enteros, muestra: suma, promedio, producto, menor y mayor. Diálogo exacto como en el libro."
  - A "STRUCTURA FINAL (copia-pegue en IntelliJ)" section shows the project tree:

```

text
Ejercicio007/
├── Ejercicio007_Main.java
├── Ejercicio007_Utilidades.java
└── Ejercicio007_Operaciones.java

```
  - A numbered list item "1. Ejercicio007\_Main.java (flujo + PROMPT)" is shown.
  - At the bottom, there's a "How can Grok help?" button and a "Copiar" (Copy) button.

**Salida de la ejecucion del programa:**

Run - Guia\_de\_ejercicios

Ejercicio007\_Main

```
C:\Users\Usuario\.jdks\openjdk-25\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.2.3\lib\idea_rt.jar=63606" -Dfile.encoding=UTF-8
```

EJERCICIO 007 - JUGANDO CON TRES NÚMEROS

Ingrese tres enteros:  
primer entero: 25  
segundo entero: 2  
tercer entero: 11

RESULTADOS

Suma: 38  
Promedio: 12.67  
Producto: 550  
Menor: 2  
Mayor: 25

Dany Noguera | dany.noguera@unah.hn | 007 | 2025-11-04

Process finished with exit code 0

## EJERCICIO#8

### Enunciado

Tomando como base el programa “Jugando con tres números” hagamos un programa que lea cuatro números, los ordene de mayor a menor y los muestre ordenados luego muestre su suma, promedio, producto, menor y mayor.

### Análisis

#### Sello Personalizado y Documentación

El **Sello Personalizado** es la VERSIÓN CORTA: Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 008 | Fecha: 2025-11-03. Se incluye en los tres archivos y se muestra al final en un cuadro. El sello garantiza la **autoría y trazabilidad**. El **prompt** completo está en **Ejercicio008\_Main.java**.

#### Programa y Modularidad

El **Programa** solicita **cuatro números** (double). Su objetivo es **ordenarlos** (mayor a menor) y calcular las cinco operaciones clave: **suma, promedio, producto, mayor y menor**.

La **Modularidad** usa tres archivos en el paquete **Ejercicio008**:

1. **Ejercicio008\_Main.java**: Flujo de control, entrada de datos y coordinación de la salida.
2. **Ejercicio008\_Utilidades.java**: Formato estético (cuadros, sello, centrado) y captura de datos.
3. **Ejercicio008\_Operaciones.java**: Lógica de cálculo (ordenamiento, suma, promedio, producto, mayor, menor).

El **Paradigma** es Estructurado, y la **Estética** usa 70 caracteres y bordes Unicode.

#### Constantes:

- **ANCHO = 70, BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '-'**

#### Subrutinas Clave (Funcionalidad):

- **Ejercicio008\_Operaciones**: ordenarMayorAMenor, calcularSuma, calcularPromedio, calcularProducto, encontrarMayor, encontrarMenor.
- **Ejercicio008\_Utilidades**: mostrarSaludo, solicitarNúmero, mostrarResultado, mostrarSelloCorto.

#### Flujo Principal (en **Ejercicio008\_Main.java**):

1. Inicia Scanner. Muestra saludo.
2. Solicita y captura los cuatro números (usando Utilidades).
3. Llama a Operaciones para realizar todos los **cálculos y el ordenamiento**.
4. Muestra los resultados (ordenados, suma, promedio, producto, mayor, menor) en cuadros (usando Utilidades).
5. Muestra el sello. Cierra Scanner.

## PROMPT

"Agente el #8 hagamos, te recuerdo nuestro sistema de archivo, en este usa los 3 que son: |— **Ejercicio008\_Main.java** (Programa principal, incluye este prompt) |— **Ejercicio008\_Utilidades.java** (Funciones de utilidad) |— **Ejercicio008\_Operaciones.java** (Lógica del ejercicio) Incluye también nuestro sello personalizado.

Necesito que realicemos el ejercicio: **Tomando como base el programa “Jugando con tres números” hagamos un programa que lea cuatro números, los ordene de mayor a menor y los muestre ordenados luego muestre su suma, promedio, producto, menor y mayor.”**

## Salida del programa

```
** RESULTADOS DE OPERACIONES **

Números Ordenados (Mayor a Menor):
12.0, 7.0, 4.0, 4.0

Suma Total: 27.00
Promedio: 6.75
Producto: 1344.00
Número Mayor: 12.00
Número Menor: 4.00

SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 008 | 2025-11-03

--- Programa finalizado con éxito ---

Proceso terminado con código de salida 0
```

## EJERCICIO#9

### Enunciado

Sobre círculo: Escriba un programa que lea el radio de un círculo como un número entero y que imprima su diámetro, circunferencia y área. Use el valor constante 3.14159 para  $\pi$ . Realice todos los cálculos en instrucciones de salida

### Análisis

El análisis del código modular del Ejercicio 001 (Cálculo del Círculo) confirma una implementación rigurosa de la Programación Estructurada (PE), destacando una clara separación de responsabilidades: la clase Operaciones se dedica exclusivamente a la lógica matemática (cálculo de diámetro, circunferencia y área, usando `Math.PI`), la clase Utilidades gestiona la presentación, incluyendo el sello personalizado con la fecha actualizada (2025-10-15) y una entrada de radio robustamente validada (asegurando un entero positivo con manejo de excepciones), y la clase Main actúa como coordinador del flujo. El requerimiento clave de realizar los cálculos en las instrucciones de salida se cumple al llamar a los métodos de Operaciones directamente dentro de las sentencias `System.out.printf()` de la clase Utilidades, manteniendo así la limpieza y la alta cohesión de cada módulo.

### PROMPT

Crea un programa en Java dentro del paquete Ejercicio009, compuesto por tres clases: Ejercicio009\_Main, Ejercicio009\_Utilidades y Ejercicio009\_Operaciones. El programa debe calcular el diámetro, la circunferencia y el área de un círculo utilizando el valor constante  $\pi=3.14159$  a partir de un radio ingresado por el usuario como número entero. Ejercicio009\_Main.java debe contener el método `main()` para coordinar la ejecución, solicitando el radio, invocando la presentación de resultados y gestionando el cierre de recursos. Ejercicio009\_Utilidades.java debe incluir métodos estáticos para: Leer el radio (`leerRadio`), asegurando que sea un entero positivo mediante manejo de excepciones (`InputMismatchException`) y bucles de validación. Mostrar el sello personalizado (`mostrarSelloCorto`) con la fecha actualizada: Cristina Cruz | Lis♥ | `cristinacruz@unah.hn` | 009 | 2025-10-15. Presentar los resultados (`mostrarResultados`), realizando una llamada directa a los métodos de la clase Operaciones en las instrucciones de salida (`System.out.printf`), sin almacenar los resultados en variables intermedias, con formato de precisión adecuada (e.g., 5 decimales para área y circunferencia). Ejercicio009\_Operaciones.java debe tener métodos estáticos para calcular las propiedades: `calcularDiametro()`, `calcularCircunferencia()`, y `calcularArea()`, y definir la constante  $\pi=3.14159$ . El programa debe ser modular, manejar estrictamente la validación del radio, mostrar una salida formateada con sello y resultados claros, y seguir las buenas prácticas de la Programación Estructurada (PE).

## PROMPT GENERADO

The screenshot shows the IntelliJ IDEA IDE. On the left, the file browser displays three files: Ejercicio009\_Main.java, Ejercicio009\_Utilidades.java, and Ejercicio009\_Operaciones.java. The main window shows the source code for Ejercicio009\_Main.java. The code includes comments indicating it was created by Cristina Cruz on October 15, 2025, and uses package Ejercicio009. It contains a main method that calls methods from Ejercicio009\_Utilidades to print a personalized stamp and calculate circle properties based on user input.

```
/*
 * Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 009 | 2025-10-15
 */
package Ejercicio009;

public class Ejercicio009_Main {
    public static void main(String[] args) {
        // Mostrar el sello personalizado
        Ejercicio009_Utilidades.mostrarSelloCorto("009", "2025-10-15");

        // Leer el radio
        int radio = Ejercicio009_Utilidades.leerRadio("Ingrese el radio (entero positivo): ");

        // Mostrar resultados
        Ejercicio009_Utilidades.mostrarResultados(radio);

        // No se requiere cerrar recursos ya que Scanner se maneja en Utilidades
    }
}
```

## Salida de la ejecución del programa:

The screenshot shows the Run tab in IntelliJ IDEA. A single run configuration named "Ejercicio009\_Main" is selected. The output window displays the program's execution. It starts with the command line used: "C:\Users\MATTA\.jdks\openjdk-25\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2.3\lib\idea\_rt.jar=60995" -Dfile.encoding=UTF-8". The program then prints its personalized stamp: "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 009 | 2025-10-15". It prompts the user for the radius of a circle: "Escriba el radio de un circulo (entero positivo): 5". The program then calculates and prints the results: "--- Resultados del Círculo (Radio = 5) ---", "Diámetro: 10.00", "Circunferencia: 31.41590", and "Área: 78.53975". Finally, it concludes with "-----" and "Programa finalizado." The process exits with an exit code of 0.

```
C:\Users\MATTA\.jdks\openjdk-25\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2.3\lib\idea_rt.jar=60995" -Dfile.encoding=UTF-8
Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 009 | 2025-10-15
Escriba el radio de un circulo (entero positivo): 5
--- Resultados del Círculo (Radio = 5) ---
Diámetro: 10.00
Circunferencia: 31.41590
Área: 78.53975
-----
Programa finalizado.

Process finished with exit code 0
```

## EJERCICIO #10

enunciado:

Sobre cuadrado: Escriba un programa que lea el lado de un cuadrado e imprima su área y perímetro antecedidos de las fórmulas usadas para calcularlos. (Se recomienda usar variables de coma flotante)

análisis:

El programa tiene como objetivo calcular e imprimir el área y el perímetro de un cuadrado, dado un valor de lado ingresado por el usuario. La entrada esperada es un único valor de coma flotante (double) que representa la longitud del lado. El programa valida el uso de fórmulas específicas: y , las cuales deben mostrarse en la salida junto al resultado. El diseño es modular, distribuido en Main (I/O principal), Operaciones (lógica de cálculo) y Utilidades (formato y mensajes). Los casos límite incluyen manejar la entrada como flotante y la complejidad algorítmica es constante () al tratarse de cálculos directos.

prompt

Crea un programa en Java que pida al usuario el lado de un cuadrado (decimal) y calcule e imprima su área y perímetro. Muestra la fórmula usada antes de cada resultado. Usa tres clases: una para la lógica, una para las utilidades de impresión y la clase principal.

prompt generado:

Para generar la solución, dividí la funcionalidad en tres partes principales.

La clase Operaciones contiene los métodos calcularArea y calcularPerimetro, los cuales reciben el valor del lado (double) y ejecutan las fórmulas y , respectivamente, devolviendo los resultados.

La clase Utilidades fue creada para manejar el formato de salida. Su método principal se encarga de recibir el nombre del cálculo, la fórmula textual, el lado y el resultado, y los imprime de forma estructurada, garantizando que la fórmula aparezca antes del valor final.

La clase Main orquesta todo. Solicita el lado al usuario, instancia Operaciones para obtener los cálculos y luego llama a Utilidades dos veces, una por cada resultado, para mostrar el área y el perímetro junto con sus fórmulas correspondientes.

## Salida:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

Advertencia: PowerShell detectó que es posible que estés usando un lector de pantalla y que hayas deshabilitado PSReadLine con fines de compatibilidad. Si quieres volver a habilitarlo, ejecuta "Import-Module PSReadLine".

PS C:\Users\erick> cd "C:\Users\erick\Documents\Proyectos Java\Guía intermedia"
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> javac -d out src\Ejercicio_010\*.java
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> java -cp out Ejercicio_010.Main
--- CÁLCULO DE ÁREA Y PERÍMETRO DE UN CUADRADO ---
Escriba la longitud del lado del cuadrado (valor en coma flotante): 3.25

--- ÁREA ---
Fórmula utilizada: Área = lado * lado
El área del cuadrado de lado 3.25 es: 10.56

--- PERÍMETRO ---
Fórmula utilizada: Perímetro = 4 * lado
El perímetro del cuadrado de lado 3.25 es: 13.00
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## EJERCICIO#11

### Enunciado

Escriba un programa que lea cinco enteros y que determine e imprima los enteros mayor y menor en el grupo.

### Análisis de Requerimientos:

- **Entrada:** Un número entero de 5 dígitos (positivo o negativo, ej. 12345 o -12345). Validar que sea entero y tenga exactamente 5 dígitos. Si no, mostrar mensaje de error y permitir reintento.
- **Procesamiento:** Extraer los 5 dígitos del número ingresado. Determinar el mayor y menor dígito usando métodos estáticos.
- **Salida:** Mostrar el número ingresado, los dígitos extraídos, y el mayor y menor dígito en cuadros elegantes (ancho 70, bordes Unicode).
- **Modularidad:**
  - Ejercicio011\_Main.java: Controla el flujo, valida entrada, y coordina operaciones y salidas.
  - Ejercicio011\_Utilidades.java: Maneja formato estético (cuadros, sello, centrado de texto).
  - Ejercicio011\_Operaciones.java: Contiene lógica para encontrar mayor y menor dígito.
- **Constantes:** ANCHO = 70, BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '\_'.
- **Estética:** Cuadros simétricos con bordes Unicode, texto centrado, diseño consistente.
- **Paradigma:** Programación estructurada, métodos estáticos, uso de Scanner.
- **Documentación:** Sello corto en cada archivo, prompt en Main.

## PROMPT

Agente tenemos que hacer el ejercicio #11 que consta de lo siguiente:

Escriba un programa que lea cinco enteros y que determine e imprima los enteros mayor y menor en el grupo.

Te recuerdo siempre muestra nuestro sello personalizado, con una salida elegante y formal y la respuesta que salga de forma elegante también junto que si la entrada no es válida el programa permita reiterar la ejecución, junto con nuestra distribución de archivos que es la siguiente para que la tengas de ejemplo permanente:

```
└── Ejercicio###/  
    ├── Ejercicio###_Main.java      → Control del flujo principal (Acá debe de ir el  
    prompt)  
    ├── Ejercicio###_Utilidades.java → Cuadros, sellos y formato estético.  
    └── Ejercicio###_Operaciones.java → Lógica matemática o funcional.
```

Y también junto que si la entrada no es válida el programa permita reiterar la ejecución.

Da un mensaje que solo permita enteros (positivos y negativos), en el caso de que el usuario ingrese decimal o un carácter o letra muestra un mensaje de entrada inválida, y muestra los números ingresados por el usuario.

## PROMPT GENERADO

Entendido, Jonatan. Aquí van los tres archivos para el Ejercicio #11 con sello, prompt en Main, cuadros simétricos (ANCHO 70, Unicode), validación de enteros (positivos/negativos, reintentos si no válido), y diseño elegante. No simulo salida, solo genero los archivos.

Ejercicio011\_Main.java  
x-java-source

Ejercicio011\_Operaciones.java  
x-java-source

Ejercicio011\_Utilidades.java  
x-java-source

Confirmo: Archivo Razonamiento más profundo en Main, ¿Cómo proceder?

How can Grok help? Auto

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 011 | 2025-10-10
 *
 * PROMPT USADO:
 * Agente tenemos que hacer el ejercicio #11 que consta de lo siguiente:
 * Escriba un programa que lea cinco enteros y que determine e imprima los enteros mayor y menor en el grupo.
 * Te recuerdo siempre muestra nuestro sello personalizado, con una salida elegante y formal y la respuesta que salga de forma elegante también junto que si la entrada no es válida el programa permita reiterar la ejecución, junto con nuestra distribución de archivos que es la siguiente para que te tengas de ejemplo permanente:
 *   Ejercicio##/
 *     Ejercicio##_.Main.java    → Control del flujo principal (Acá debe de ir el prompt)
 *     Ejercicio##_.Utilidades.java → Cuadros, sellos y formato estético.
 *     Ejercicio##_.Operaciones.java → Lógica matemática o funcional.
 * Y también junto que si la entrada no es válida el programa permita reiterar la ejecución.
 * Da un mensaje que solo permita enteros (positivos y negativos), en el caso de que el usuario ingrese decimal o un carácter o letra muestra un mensaje de entrada inválida, y muestra los números ingresados por el usuario.
 */
package Ejercicio011;

import java.util.Scanner;

public class Ejercicio011_Main {
    private static final String LINEA = "=";
    private static final int ANCHO = 70;
```

## Salida de la ejecución del programa:

```
Ingrese el número: 50123
Número ingresado: 50123

DÍGITOS INGRESADOS
Digitos: 5, 0, 1, 2, 3

RESULTADOS
Mayor digito: 5
Menor digito: 0

SELLO PERSONALIZADO - VERSIÓN CORTA
Jonatan Varela | jivarelag@unah.hn | 011 | 2025-10-10

*** Operaciones completadas exitosamente ***

Process finished with exit code 0
```

## EJERCICIO#12

### Enunciado

Haremos uso del conocimiento producto del desarrollo del ejercicio “*Par o impar*” para hacer un programa que lea cinco números, al final imprimirá los números ordenados de menor a mayo y dirá cuántos son pares y cuántos son impares.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 012 | 2025-11-04) dentro de los tres archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Solicitar Cinco números enteros ingresados por el usuario, Almacenar los números en un arreglo. Ordenarlos de menor a mayor. Contar cuántos son pares y cuántos impares. Mostrar Lista ordenada de números (menor → mayor). Cantidad de números pares. Cantidad de números impares.
- **Modularidad:** Tres archivos en paquete Ejercicio012:
  - Ejercicio012\_Main.java: Flujo principal del programa. Pide datos al usuario, llama métodos de operaciones, muestra resultados usando utilidades; incluye el prompt.
  - Ejercicio012\_Utilidades.java: Estética: encabezados, separadores, centrado de texto, sello corto.
  - Ejercicio012\_Operaciones.java: Lógica: ordenar el arreglo, contar pares e impares. No imprime nada.
- **Paquete:** Ejercicio012.
- **Estética:** Bordes Unicode (Γ, Λ, Τ, Σ, —, |) para un diseño simétrico y elegante. Texto centrado, Títulos y separadores decorados, Sello corto al final del programa.
- **Paradigma:** Programación Estructurada, métodos estáticos y flujo secuencial..
- **Documentación:** - Comentarios tipo encabezado (sello completo y versión corta).
  - Comentarios dentro del código explicando lo importante.
  - No se documenta lógica en Operaciones más allá de Javadoc simple.

### **Subrutinas (en Ejercicio012\_Utilidades):**

- Recibe textos para encabezados, separadores o pie.
- Genera bordes superiores o inferiores con caracteres Unicode.
- Centra el texto dentro del ancho definido.
- Imprime en consola.

### **Subrutinas (en Ejercicio012\_Operaciones):**

- Recibe los cinco números como parámetros.  
(procesa datos: ordena, cuenta pares/impares)
- Retorna el resultado.
- No imprime nada en consola.

### **Flujo Principal (en Ejercicio012\_Main):**

- Orquesta todo el proceso:
  - muestra encabezado, solicita números, llama a operaciones matemáticas, imprime resultados con formato visual y finaliza con pie de ejecución.

### **PROMPT**

Ejercicio #12: Haremos uso del conocimiento producto del desarrollo del ejercicio “Par o impar” para hacer un programa que lea cinco números, al final imprimirá los números ordenados de menor a mayor y dirá cuántos son pares y cuántos son impares. te recuerdo que en todo deben de ir el sello corto anteriormente hecho, y nuestra distribución de 3 archivos. y además me desglose los requerimientos llenando los campos siguientes: - Sello personalizado corto, - Requerimientos del programa, - Modularidad, - Estética, - Paradigma, - Documentación.

## PROMPT GENERADO

The screenshot shows a code editor window titled "Ejercicio012\_Main.java". The code is a Java program that reads five integers from the user, sorts them, and counts the number of even and odd numbers. It includes developer metadata at the top and a copyright notice at the bottom. The code uses standard Java syntax with imports for Scanner and the package declaration.

```
java

// | _____ Metadatos del Desarrollador
// | Nombre: Dany Noguera
// | Correo: dany.noguera@unah.hn
// | Fecha: 2025-11-04
// | Código del Ejercicio: 012
// | Enunciado: Programa que lee cinco números, los ordena y cuenta cuántos
// | son pares e impares.
// | _____
// Versión corta: Dany Noguera | dany.noguera@unah.hn | 012 | 2025-11-04

package Ejercicio012;
import java.util.Scanner;

public class Ejercicio012_Main {
    private static final int ANCHO = 78;
    public static void main(String[] args) {
```

## Salida de la ejecución del programa:

The screenshot shows the IntelliJ IDEA run console for the "Ejercicio012\_Main" run configuration. The console output shows the program's execution steps, including the developer's name and date, the input of five integers (5, 1, 2, 3, 4), the sorted output ([1, 2, 3, 4, 5]), and the count of even (2) and odd (3) numbers. The output concludes with the developer's information and a completion message.

```
EJERCICIO 012 - ORDENANDO Y CONTANDO PARES/IMPARES

Ingrese cinco números enteros:
Número 1: 5
Número 2: 1
Número 3: 2
Número 4: 3
Número 5: 4

RESULTADOS

Números ordenados: [1, 2, 3, 4, 5]
Cantidad de pares: 2
Cantidad de impares: 3

Dany Noguera | dany.noguera@unah.hn | 012 | 2025-11-04

;Ejercicio completado!
;Ejercicio 012 completado!
```

## EJERCICIO#13

### Enunciado

**Par o impar:** Escriba un programa que lea un entero y que determine e imprima si es impar o par

### Análisis

#### Sello Personalizado y Documentación

El Sello Personalizado es la VERSIÓN CORTA: Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 013 | Fecha: 2025-11-03. Se incluye en los dos archivos y se muestra al final en un cuadro. El sello garantiza la autoría y trazabilidad. El prompt completo está en Ejercicio013\_Main.java.

#### Programa y Modularidad

El Programa solicita un único número entero (int) al usuario. Su objetivo es determinar e imprimir si dicho número es par o impar utilizando el operador módulo (%).

La Modularidad usa dos archivos en el paquete Ejercicio013:

1. Ejercicio013\_Main.java: Flujo de control, entrada de datos, lógica de Par/Impar, y coordinación de la salida.
2. Ejercicio013\_Utilidades.java: Formato estético (cuadros, sello, centrado) y captura de datos formateada.

El Paradigma es Estructurado, y la Estética usa 70 caracteres de ancho y bordes Unicode.

#### Constantes:

- **ANCHO = 70, BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '-'**

#### Subrutinas Clave:

- **Ejercicio013\_Utilidades:** mostrarSaludo, solicitarEntero, mostrarResultado, mostrarSelloCorto.

### Flujo Principal (en Ejercicio013\_Main.java):

1. Inicia Scanner. Muestra saludo.
2. Solicitud y captura el número entero (usando Utilidades).
3. Aplica la **Lógica**: numero % 2 == 0 para determinar si es "PAR" o "IMPAR".
4. Muestra el resultado (número ingresado y su clasificación) en un cuadro (usando Utilidades).
5. Muestra el sello. Cierra Scanner.

### PROMPT

"Agente el #8 hagamos, te recuerdo nuestro sistema de archivo, en este usa los 3 que son: |— **Ejercicio008\_Main.java** (Programa principal, incluye este prompt) |— **Ejercicio008\_Utilidades.java** (Funciones de utilidad) |— **Ejercicio008\_Operaciones.java** (Lógica del ejercicio) Incluye también nuestro sello personalizado.

Necesito que realicemos el ejercicio: **Par o impar: Escriba un programa que lea un entero y que determine e imprima si es impar o par (Ejercicio 2.24 Libro 1).**"

Salida	del	programa
<pre>INGRESO DEL NÚMERO</pre>		
<pre>Ingrese un número entero para determinar si es Par o Impar: &gt;&gt; Ingrese valor: 3</pre>		
<pre>** RESULTADO DEL ANÁLISIS **</pre>		
<pre>El número 3 es: IMPAR</pre>		
<pre>SELLO PERSONALIZADO</pre>		
<pre>Kevin Lainez   klainezs@unah.hn   013   2025-11-03</pre>		

## EJERCICIO#14

### Enunciado

Tablero de ajedrez: Escriba una aplicación que muestre un patrón de tablero de damas con ocho instrucciones de salida, y después muestre el mismo patrón utilizando el menor número de instrucciones posible.

### Análisis

El análisis del código del Ejercicio 014 (Tablero de Ajedrez) confirma su adhesión al modelo de Programación Estructurada (PE), aunque la lógica es primariamente de formato y salida. La modularidad se mantiene con Ejercicio014\_Operaciones definiendo una constante para la casilla (\* ), Ejercicio014\_Utilidades manejando toda la lógica de impresión y el sello, y Ejercicio014\_Main coordinando el flujo. El código cumple perfectamente con el requerimiento de demostrar la impresión del tablero de dos maneras: primero, utilizando ocho instrucciones de salida (System.out.println), lo que es didáctico; y segundo, utilizando la mínima cantidad de instrucciones posible (sólo una instrucción System.out.printf), lo cual demuestra eficiencia y el poder de las funciones de formato y repetición (String.repeat()), manteniendo la separación de responsabilidades y el sello (2025-10-15) intacto en la documentación.

### PROMPT

Creame un código modular del Ejercicio 014 (Tablero de Ajedrez) exige un programa en Java dentro del paquete Ejercicio014 que demuestra dos formas de imprimir un patrón de tablero de 8x8 usando el carácter "\* ", con una estructura estrictamente de Programación Estructurada (PE) dividida en tres módulos: Ejercicio014\_Operaciones (definiendo solo la constante de la casilla), Ejercicio014\_Utilidades (manejando el sello Cristina Cruz... 2025-10-15, y conteniendo las funciones de impresión), y Ejercicio014\_Main (coordinando la ejecución). El programa debe cumplir el doble requerimiento de mostrar el patrón primero mediante ocho instrucciones de salida separadas para fines didácticos, y luego utilizando la mínima cantidad de instrucciones de salida posible (idealmente una única instrucción) para demostrar eficiencia, todo mientras mantiene el sello personalizado como documentación en los tres archivos y lo muestra al inicio.

## PROMPT

correcta, pero la repito aquí para confirmar que incluye el paquete y está bien estructurada:

```
</> Ejercicio004_Utilidades.java  
x-java-source
```

Editar en archivos • Mostrar esquemas alineados

**Confirmación:** Este archivo es idéntico al que proporcionaste y está correcto.

Ejercicio004\_Operaciones.java

Tu versión de Ejercicio004\_Operaciones.java también es correcta:

```
</> Ejercicio004_Operaciones.java  
x-java-source
```

Editar en archivos • Mostrar esquemas alineados

**Confirmación:** Este archivo es idéntico al que proporcionaste y está correcto.

How can Grok help? Auto ▾

## GENERADO

```
/*  
 * Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 014 | 2025-10-15  
 */  
package Ejercicio014;  
  
public class Ejercicio014_Main {  
    public static void main(String[] args) {  
        // Mostrar el sello personalizado  
        Ejercicio014_Utilidades.mostrarSelloCorto("014", "2025-10-15");  
  
        System.out.println("Patrón del tablero de ajedrez (ocho instrucciones separadas):");  
        Ejercicio014_Utilidades.imprimirTableroOchoInstrucciones();  
  
        System.out.println("\nPatrón del tablero de ajedrez (mínima cantidad de instrucciones):");  
        Ejercicio014_Utilidades.imprimirTableroEficiente();  
    }  
}
```

## Salida de la ejecución del programa:



## EJERCICIO #15

### **enunciado:**

Figuras: Escriba un programa que imprima un cuadro, un óvalo, una flecha y un diamante como se muestra a continuación (Ejercicio 2.21 Libro 1):

### **análisis:**

El objetivo de este programa es generar cuatro figuras geométricas básicas (cuadro, óvalo, flecha, diamante) utilizando caracteres ASCII (\*, espacios) en la consola. No requiere entradas de usuario, por lo que las operaciones son de tipo I/O puramente. La salida es una secuencia fija de impresiones en pantalla, con separadores visuales para mejorar la claridad. El código está modularizado en Operaciones (lógica de dibujo), Utilidades (separadores) y Main (orquestación del flujo). No hay casos límite ni validaciones de entrada, ya que la salida es predefinida y constante. La complejidad algorítmica es (constante).

### **prompt**

Crea un programa en Java que imprima en la consola un cuadro, un óvalo, una flecha y un diamante, usando solo caracteres de asterisco (\*) y espacios. Cada figura debe estar separada por una línea visual. Modulariza la solución en una clase para los dibujos, una para las utilidades y la clase principal.

### **prompt generado:**

Para generar este programa, la estrategia fue dividir el trabajo de impresión.

La clase Operaciones contiene cuatro métodos específicos: imprimirCuadro, imprimirOvalo, imprimirFlecha e imprimirDiamante. Cada uno de estos métodos utiliza una secuencia de sentencias System.out.println para generar la forma deseada con caracteres fijos en la consola.

La clase Utilidades es simple y se usa para el formato, conteniendo solo un método estático (imprimirSeparador) que imprime una línea de guiones.

Finalmente, la clase Main actúa como el orquestador, instancia la clase Operaciones y llama a cada uno de sus métodos de impresión de forma secuencial, intercalando las llamadas a Utilidades.imprimirSeparador() entre cada figura para lograr el espacio visual requerido.

### **Salida:**

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

Advertencia: PowerShell detectó que es posible que estés usando un lector de pantalla y que hayas deshabilitado PSReadLine con fines de compatibilidad. Si quieres volver a habilitarlo, ejecuta "Import-Module PSReadLine".

PS C:\Users\erick> cd "C:\Users\erick\Documents\Proyectos Java\Guía intermedia"
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> javac -d out src\Ejercicio_010/*.java
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> java -cp out Ejercicio_010.Main
--- CÁLCULO DE ÁREA Y PERÍMETRO DE UN CUADRADO ---
Escriba la longitud del lado del cuadrado (valor en coma flotante): 3.25

--- ÁREA ---
Fórmula utilizada: Área = lado * lado
El área del cuadrado de lado 3.25 es: 10.56

--- PERÍMETRO ---
Fórmula utilizada: Perímetro = 4 * lado
El perímetro del cuadrado de lado 3.25 es: 13.00
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## EJERCICIO#16

### ENUNCIADO

Split: Escriba un programa que reciba como entrada un número entero (que incluyen los positivos y los negativos, los decimales no) de cinco dígitos, que separe ese número en sus dígitos individuales y los imprima, cada uno separado de los demás por tres espacios. [Sugerencia: use los operadores de división entera y módulo.] Por ejemplo, si el usuario escribe el número 42339, el programa debe imprimir: 4 2 3 3 9

### Análisis de Requerimientos

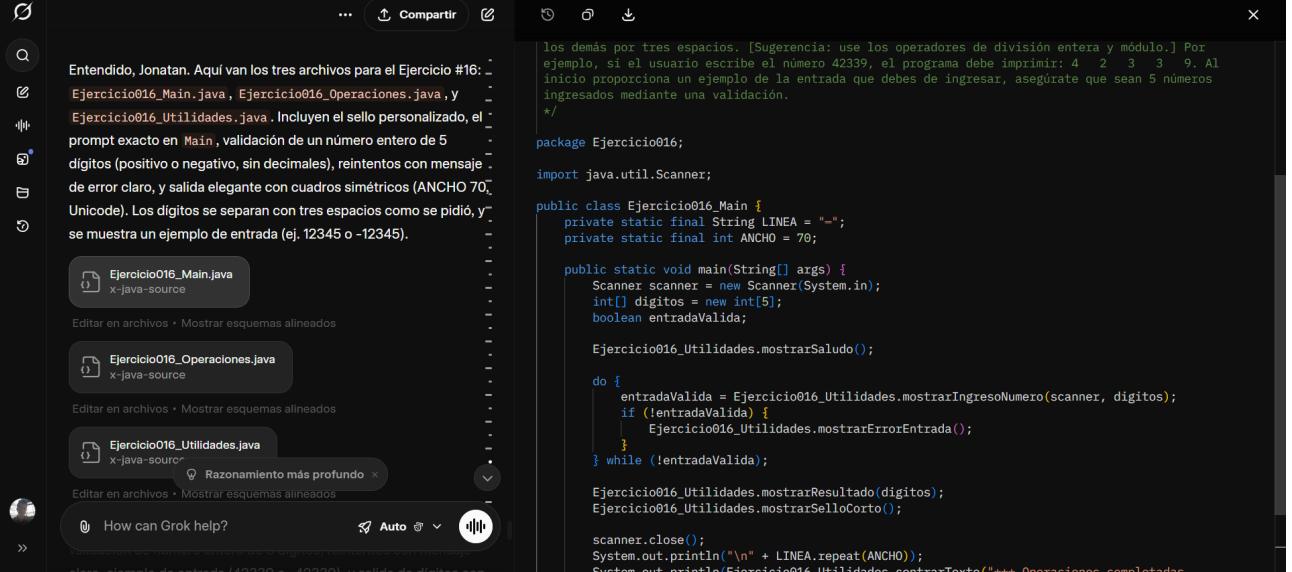
- **Entrada:** Un número entero de 5 dígitos (positivo o negativo, ej. 42339 o -42339). Validar que sea entero y tenga exactamente 5 dígitos. Si no, mostrar mensaje de error y permitir reintento.
- **Procesamiento:** Extraer los 5 dígitos del número usando división entera y módulo. Almacenar en un arreglo para su impresión.
- **Salida:** Mostrar el número ingresado y los dígitos separados por tres espacios (ej. 4 2 3 3 9) en cuadros elegantes (ancho 70, bordes Unicode).
- **Modularidad:**
  - Ejercicio016\_Main.java: Controla el flujo, valida entrada, y coordina operaciones y salidas.
  - Ejercicio016\_Utilidades.java: Maneja formato estético (cuadros, sello, centrado de texto, validación con reintentos).
  - Ejercicio016\_Operaciones.java: Contiene lógica para separar dígitos usando división y módulo.
- **Constantes:** ANCHO = 70, BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '\_'.
- **Estética:** Cuadros simétricos con bordes Unicode, texto centrado, diseño consistente.
- **Paradigma:** Programación estructurada, métodos estáticos, uso de Scanner.
- **Documentación:** Sello corto en cada archivo, prompt exacto en Main.

### PROMPT

Agente nuevo ejercicio a realizar el #16, te recuerdo que debe de ir nuestro sello personalizado, la capacidad de reiterar el ejercicio si la entrada es invalida, con mensajes de salida simples para el usuario, junto con nuestra distribución de archivos, el ejercicio es: Split: Escriba un programa que reciba como entrada un número entero (que incluyen los positivos y los negativos los decimales no) de cinco dígitos, que separe ese número en sus dígitos individuales y los imprima, cada uno separado de los demás por tres espacios. [Sugerencia: use los operadores de división entera y módulo.] Por ejemplo, si el usuario escribe el número 42339, el programa debe imprimir: 4 2 3 3 9. Al inicio proporciona un ejemplo de la entrada

que debes de ingresar, asegúrate que sean 5 números ingresados mediante una validación.

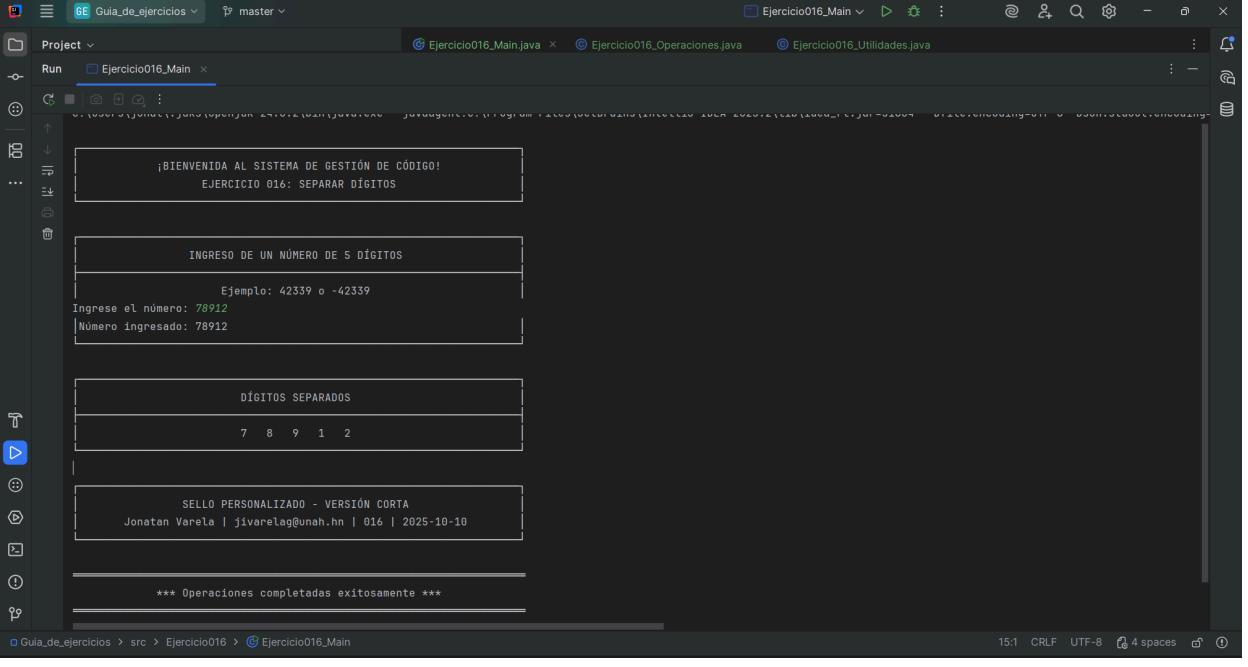
## PROMPT GENERADO



```
Entendido, Jonatan. Aquí van los tres archivos para el Ejercicio #16:  
Ejercicio016_Main.java, Ejercicio016_Operaciones.java, y  
Ejercicio016_Utilidades.java. Incluyen el sello personalizado, el  
prompt exacto en Main, validación de un número entero de 5  
dígitos (positivo o negativo, sin decimales), reintentos con mensaje  
de error claro, y salida elegante con cuadros simétricos (ANCHO 70,  
Unicode). Los dígitos se separan con tres espacios como se pidió, y  
se muestra un ejemplo de entrada (ej. 12345 o -12345).
```

```
los demás por tres espacios. [Sugerencia: use los operadores de división entera y módulo.] Por  
ejemplo, si el usuario escribe el número 42339, el programa debe imprimir: 4 2 3 3 9. Al  
inicio proporciona un ejemplo de la entrada que debes de ingresar, asegúrate que sean 5 números  
ingresados mediante una validación.  
*/  
  
package Ejercicio016;  
  
import java.util.Scanner;  
  
public class Ejercicio016_Main {  
    private static final String LINEA = "=";  
    private static final int ANCHO = 70;  
  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        int[] digitos = new int[5];  
        boolean entradaValida;  
  
        Ejercicio016_Utilidades.mostrarSaludo();  
  
        do {  
            entradaValida = Ejercicio016_Utilidades.mostrarIngresoNumero(scanner, digitos);  
            if (!entradaValida) {  
                Ejercicio016_Utilidades.mostrarErrorEntrada();  
            }  
        } while (!entradaValida);  
  
        Ejercicio016_Utilidades.mostrarResultado(digitos);  
        Ejercicio016_Utilidades.mostrarSelloCorto();  
  
        scanner.close();  
        System.out.println("\n" + LINEA.repeat(ANCHO));  
        System.out.println(Ejercicio016_Utilidades.centrarTexto("*** Operaciones completadas"));  
    }  
}
```

## Salida de la ejecución del programa:



```
¡BIENVENIDA AL SISTEMA DE GESTIÓN DE CÓDIGO!  
EJERCICIO 016: SEPARAR DÍGITOS  
  
INGRESO DE UN NÚMERO DE 5 DÍGITOS  
Ejemplo: 42339 o -42339  
Ingrese el número: 78912  
Número ingresado: 7 8 9 1 2  
  
DÍGITOS SEPARADOS  
7 8 9 1 2  
  
SELLO PERSONALIZADO - VERSIÓN CORTA  
Jonatan Varela | jivarelag@unah.hn | 016 | 2025-10-10  
  
*** Operaciones completadas exitosamente ***
```

## EJERCICIO#17

### Enunciado

Tomando como base el ejercicio “par o impar” ahora usted lo que debe hacer es agarrar ese número y mostrar gráficas horizontales de la siguiente manera: 1. tantos asteriscos como unidades posea el número. 2. Cada asterisco de la gráfica representa dos unidades. 3. Cada asterisco de la gráfica representa cinco unidades.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 017 | 2025-11-04) dentro de los tres archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Solicitar un número entero ingresado por el usuario., Almacenar los números en un arreglo. Determinar si es par o impar (retomar ejercicio previo).
  - Generar 3 gráficas horizontales usando \*:
    - 1) Cada \* = 1 unidad
    - 2) Cada \* = 2 unidades
    - 3) Cada \* = 5 unidades.
      - Menú de selección de idioma (Español / Inglés).
      - Segundo menú para ejecutar o salir.
- **Modularidad:** Cuatro archivos en paquete Ejercicio017:
  - Ejercicio017\_Main.java: Flujo principal del programa. Pide datos al usuario, llama métodos de operaciones, muestra resultados usando utilidades; incluye el prompt.
  - Ejercicio017\_Menus.java: Menus (Idioma y confirmacion).
  - Ejercicio017\_Utilidades.java: Estética: encabezados, separadores, centrado de texto, sello corto.
  - Ejercicio017\_Operaciones.java: Lógica: ordenar el arreglo, contar pares e impares. No imprime nada.
- **Paquete:** Ejercicio017.
- **Estética:** Bordes Unicode (Γ, Λ, Τ, Σ, –, |) para un diseño simétrico y elegante. Debe mostrar encabezados, separadores y sello corto final.
- **Paradigma:** Programación estructurada (métodos estáticos, sin POO).
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

### **Subrutinas (en Ejercicio017\_Menu):**

- menul idioma(Scanner sc) 1. Muestra un menú de idiomas (Español / Inglés).  
2. Espera una opción válida (1 o 2).  
3. Si la opción es incorrecta, muestra mensaje de error y repite.  
4. Devuelve el idioma seleccionado (1 = Español, 2 = Inglés).
- menuAcción(Scanner sc, int idioma) 1. Muestra un menú dependiendo del idioma: **Ejecutar / Salir**.  
2. Espera una opción válida (1 o 2).  
3. Si la opción no es válida, indica error y repite.  
4. Si el usuario elige **Ejecutar**, devuelve **true**.  
5. Si elige **Salir**, devuelve **false**.

### **Subrutinas (en Ejercicio017\_Utilidades):**

- mostrarEncabezado() Muestra sello y título del ejercicio.
- mostrarSeparador() Separa secciones visualmente.
- centrarTexto() Centra texto dentro del marco.
- mostrarVersionCorta() Sello corto final.
- mostrarPie() Mensaje final del programa.

### **Subrutinas (en Ejercicio017\_Operaciones):**

- esPar(int num) Determina si el número es par o impar.
- graficaUno(int num) Genera gráfica usando num asteriscos.
- graficaDos(int num) Genera gráfica usando num / 2 asteriscos.
- graficaCinco(int num) Genera gráfica usando num / 5 asteriscos.

### **Flujo Principal (en Ejercicio017\_Main):**

- main() Control secuencial del programa.
- menul idioma() Muestra menú de idioma y retorna selección.
- menuAcción() Pregunta si ejecutar o salir.

### **PROMPT**

A partir de acá todos los ejercicios contenidos en esta guía deberán contar con un menú de selección de idioma y luego con un segundo menú donde pueda elegirse entre ejecutarse o interrumpir la ejecución (finalizar el programa).

17. Tomando como base el ejercicio “par o impar” ahora usted lo que debe hacer es agarrar ese número y mostrar gráficas horizontales de la siguiente manera:

1. tantos asteriscos como unidades posea el número.
2. Cada asterisco de la gráfica representa dos unidades.
3. Cada asterisco de la gráfica representa cinco unidades.

Por ejemplo si se lee el 15:

Número 15 es impar:

15 | \* \* \* \* \* \* \* \* \* \* \* \* (15 asteriscos, cada uno vale 1)

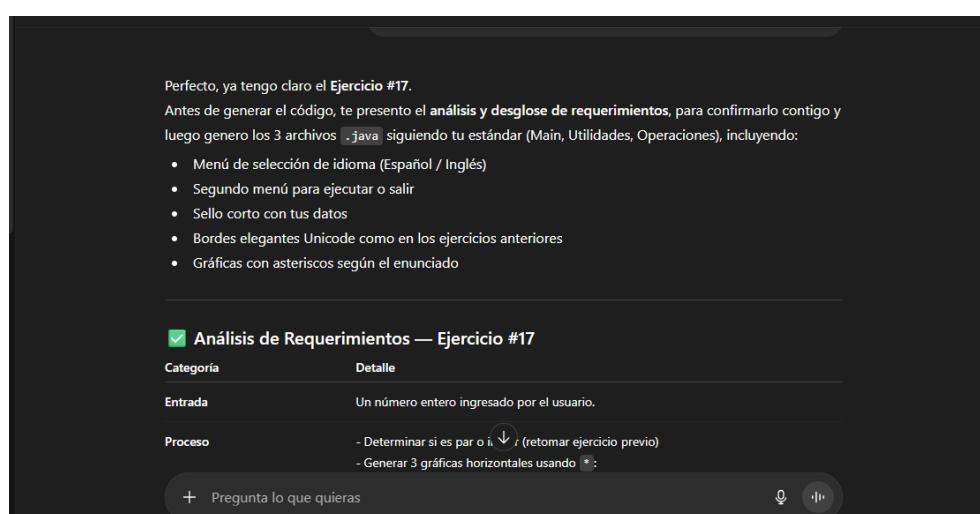
15 | \* \* \* \* \* \* (8 asteriscos, cada uno vale 2)

15 | \* \* \* (3 asteriscos, cada uno vale 5).

Además necesito distribuir los dos nuevos menús en un nuevo archivo llamado:

Menus.java para una mejor modularidad

## PROMPT GENERADO



## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run Ejercicio017_Main x
C S E : 
EJERCICIO 017 - GRÁFICAS HORA
Seleccione idioma / Select language:
1. Español
2. English
> 2

Select an option:
1. Run program
2. Exit
> 1
Enter an integer number: 8

RESULTS
Number 8 is EVEN.

8 | * * * * * * * (value 1)
8 | * * * * (value 2)
8 | * (value 5)

Select an option:
1. Run program
2. Exit
```

## EJERCICIO#18

### Enunciado

ellos les gusta la gasolina: Los conductores se preocupan acerca del kilometraje de sus automóviles. Un conductor ha llevado el registro de varios reabastecimientos de gasolina, registrando los kilómetros conducidos y los litros usados en cada reabastecimiento. Desarrolle un programa en JAVA que utilice una instrucción while para recibir como entrada los kilómetros conducidos y los litros usados (ambos como enteros) por cada reabastecimiento. El programa debe calcular y mostrar los kilómetros por litro obtenidos en cada reabastecimiento, y debe imprimir el total de kilómetros por litro obtenidos en todos los reabastecimientos hasta este punto

### Análisis

#### Sello Personalizado y Documentación

El Sello Personalizado es la VERSIÓN CORTA: Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 018 | Fecha: 2025-11-03. Se incluye como comentario de bloque en la cabecera de los tres archivos y se muestra en un cuadro estético al final de la ejecución. El prompt completo está en Ejercicio018\_Main.java.

#### Programa y Modularidad

El Programa debe leer una serie de pares de enteros: kilómetros conducidos y litros usados para cada reabastecimiento. El flujo debe usar un ciclo while que termina con un valor centinela (ej., kilómetros = -1). El sistema debe calcular y mostrar el KPL (Kilómetros por Litro) por reabastecimiento y el KPL total acumulado.

La Modularidad usa tres archivos en el paquete Ejercicio018:

1. Ejercicio018\_Main.java: Flujo de control, maneja el ciclo while y la entrada del centinela.
2. Ejercicio018\_Utilidades.java: Formato estético, cuadros, sello y funciones de entrada/salida para solicitar valores y mostrar resultados parciales y finales.
3. Ejercicio018\_Operaciones.java: Lógica de cálculo (calcularKPLReabastecimiento, calcularKPLTotal) y manejo de totales acumulados.

El Paradigma es Estructurado, y la Estética usa 70 caracteres de ancho y bordes Unicode.

**Constantes:**

- ANCHO = 70, BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '-'

**Subrutinas Clave (Funcionalidad):**

- Ejercicio018\_Operaciones: tirar (simula el lanzamiento, devuelve 0 para Cara, 1 para Cruz).
- Ejercicio018\_Utilidades: mostrarSaludo, solicitarValor (para kilómetros y litros), mostrarResultadosParciales, mostrarResumenFinal, mostrarSelloCorto.

**Flujo Principal (en Ejercicio018\_Main.java):**

1. Inicia Scanner. Muestra saludo e instrucciones (centinela: -1).
2. Inicializa un objeto de Operaciones para llevar los totales acumulados (km total, litros total).
3. Inicia el ciclo while: a. Sigue Kilómetros (si es -1, termina). b. Sigue Litros. c. Llama a Operaciones para calcular el KPL actual y acumular los totales. d. Muestra el KPL actual y el KPL total acumulado (usando Utilidades).
4. Al salir del ciclo, muestra un resumen final de los totales generales.
5. Muestra el sello. Cierra Scanner.

**PROMPT**

"Agente el #18 hagamos, te recuerdo nuestro sistema de archivo, en este usa los 3 que son: |— Ejercicio018\_Main.java (Programa principal, incluye este prompt) |— Ejercicio018\_Utilidades.java (Funciones de utilidad) |— Ejercicio018\_Operaciones.java (Lógica del ejercicio) Incluye también nuestro sello personalizado.

Necesito que realicemos el ejercicio: A ellos les gusta la gasolina: Los conductores se preocupan acerca del kilometraje de sus automóviles. Un conductor ha llevado el registro de varios reabastecimientos de gasolina, registrando los kilómetros conducidos y los litros usados en cada reabastecimiento. Desarrolle un programa en JAVA que utilice una instrucción while para recibir como entrada los kilómetros conducidos y los litros usados (ambos como enteros) por cada reabastecimiento. El programa debe calcular y mostrar los kilómetros por litro obtenidos en cada reabastecimiento, y debe imprimir el total de kilómetros por litro obtenidos en todos los reabastecimientos hasta este punto (Ejercicio 4.13 Libro 1)."

**Salida del programa**

INICIO DE REGISTRO DE REABASTECIMIENTOS

Kilómetros recorridos (o -1 para terminar):

>> Ingrese valor: 1

Litros usados:

>> Ingrese valor: 2

Reabastecimiento #1:

KPL Actual: 0.50 km/l

KPL Total Acumulado: 0.50 km/l

Siguientes kilómetros (o -1 para terminar):

>> Ingrese valor: -1

--- RESUMEN FINAL ---

Total Kilómetros Recorridos: 1.00 km

Total Litros Consumidos: 2.00 l

\*\* KPL PROMEDIO FINAL: 0.50 km/l \*\*

SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 018 | 2025-11-03

## EJERCICIO#19

### Enunciado

Gasto lo que no tengo: Desarrolle una aplicación en JAVA que determine si alguno de los clientes de una tienda de departamentos se ha excedido del límite de crédito en una cuenta. Para cada cliente se tienen los siguientes datos:

- A. Número de cuenta (un entero)
- B. Saldo al inicio del mes
- C. Total de todos los artículos cargados por el cliente en el mes
- D. Total de todos los créditos aplicados a la cuenta del cliente en el mes
- E. Límite de crédito permitido.

El programa debe usar una instrucción while para recibir como entrada cada uno de estos datos, debe calcular el nuevo saldo (= saldo inicial + cargos – créditos) y determinar si este excede el límite de crédito del cliente. Para los clientes cuyo límite de crédito sea excedido, el programa debe mostrar el número de cuenta del cliente, su límite de crédito, el nuevo saldo y el mensaje "Se excedió el límite de su crédito"

### Análisis

El código del Ejercicio 019 (Límite de Crédito) logra una modularidad avanzada en la Programación Estructurada (PE) al incorporar la internacionalización y el control de flujo inicial mediante menús, además de la lógica de negocio. La clase Ejercicio019\_Utilidades actúa como el núcleo de la interfaz y la lógica de control, gestionando un diccionario multilingüe (textos) para los menús de idioma y ejecución, y proporcionando métodos robustos de validación (leerEntero, leerDouble) que manejan errores de tipo (InputMismatchException). Por otro lado, la clase Ejercicio019\_Operaciones mantiene la lógica pura de negocio, calculando el nuevo saldo y verificando el límite; finalmente, Ejercicio019\_Main coordina el flujo secuencial del programa mediante un bucle while que procesa múltiples clientes, y utiliza los códigos numéricos del diccionario de Utilidades para asegurar que todas las interacciones con el usuario, incluida la advertencia de excedente de crédito, se muestran en el idioma seleccionado.

### PROMPT

Creame un código para el Ejercicio 019 (Gasto lo que no tengo) exige una aplicación en Java, dentro del paquete Ejercicio019, que procese cuentas de clientes para determinar si excedieron su límite de crédito. La aplicación debe comenzar con un menú de selección de idioma y luego un menú de ejecución (Ejecutar/Finalizar). La estructura debe ser de Programación Estructurada (PE), dividida en Ejercicio019\_Operaciones (para calcular el nuevo saldo = saldo inicial + cargos – créditos, y verificar el límite), Ejercicio019\_Utilidades (para gestionar los menús multilingües, la entrada validada de datos double/int, y la presentación de la advertencia si el límite es excedido), y Ejercicio019\_Main (coordinando el flujo

mediante un bucle while que procese clientes hasta que se ingrese cuenta 0). El programa debe mostrar el número de cuenta, límite de crédito, nuevo saldo y el mensaje "Se excedió el límite de su crédito" solo si aplica, manteniendo el sello (Cristina Cruz... 2025-10-15) en la documentación de los tres archivos.

## PROMPT GENERADO

The image shows a screenshot of a Java code editor. On the left, there is a sidebar titled "Si Usas un IDE" with instructions for creating a project, adding files, verifying the package, executing, and checking the console. A "Grok" interface is overlaid on the sidebar, showing a question "How can Grok help?", a dropdown menu set to "Auto", and a microphone icon. On the right, the Java code for "Ejercicio019\_Main.java" is displayed:

```
package Ejercicio019;
import java.util.Scanner;

/*
 * Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 019 | 2025-10-15
 */
public class Ejercicio019_Main {

    private static final int MSG_CUENTA = 11;
    private static final int MSG_SALDO_INICIAL = 12;
    private static final int MSG_CARGOS = 13;
    private static final int MSG_CREDITOS = 14;
    private static final int MSG_LIMITE = 15;

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

        Ejercicio019_Utilidades.mostrarSelloCorto();

        Ejercicio019_Utilidades.seleccionarIdioma(scanner);

        if (!Ejercicio019_Utilidades.mostrarMenuEjecucion(scanner)) {
            scanner.close();
            return;
        }

        Ejercicio019_Utilidades.mostrarEncabezado();

        int numeroCuenta = Ejercicio019_Utilidades.leerEntero(scanner,
            "Introduzca el número de cuenta: ");
    }
}
```

```
Run Ejercicio019_Main <-->

↑ | Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 019 | 2025-10-15 |
↓
≡↔ Seleccione el idioma:
≡↓ [1] Español
≡↓ [2] Inglés
≡ 1

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>1

--- Análisis de Crédito de Clientes ---
Ingrese el número de cuenta (o 0 para finalizar): 2023
Ingrese el saldo al inicio del mes: 25000
Ingrese el total de cargos del mes: 15000
Ingrese el total de créditos aplicados: 5
Ingrese el límite de crédito: 10
-----
Cuenta: 2023
Límite de crédito: $10.00
Nuevo saldo: $39995.00
Se excedió el límite de su crédito
```

**Salida de la ejecución del programa:**

## EJERCICIO #20

### enunciado:

Hora de pagar: Una gran empresa de químicos paga a sus vendedores mediante comisiones. Los vendedores reciben \$200 por semana, más el 9% de sus ventas brutas durante esa semana. Por ejemplo, un vendedor que vende \$5000 de mercancía en una semana, recibe \$200 más el 9% de \$5000, o un total de \$650. Desarrolle un programa en JAVA que utilice una instrucción while para recibir como entrada las ventas brutas de cada vendedor de la semana anterior, y que calcule y muestre los ingresos de ese vendedor. Procese las cifras de un vendedor a la vez (Ejercicio 4.15 Libro 1).

### análisis:

El propósito de este programa es calcular los ingresos semanales de una cantidad indeterminada de vendedores, aplicando la fórmula . La entrada esperada son las ventas brutas de cada vendedor como un valor de coma flotante. El procesamiento se realiza secuencialmente mediante un ciclo while que termina cuando el usuario introduce un valor centinela (cero o negativo). El diseño es altamente modular: Operaciones contiene la lógica de cálculo, Utilidades gestiona los textos y el soporte bilingüe (ES/EN), y Menus maneja la navegación. Se incluye manejo de errores básico para entradas inválidas. La complejidad del cálculo es , mientras que el ciclo de procesamiento es , donde es el número de vendedores.

### prompt

Necesito un programa en Java que calcule los ingresos semanales de vendedores. Cada vendedor gana \$200 fijos más el 9% de comisión sobre sus ventas brutas. Usa un ciclo while para solicitar las ventas de cada vendedor y finalizar la entrada cuando se ingrese un valor . La solución debe ser modular con soporte para español e inglés, y un menú de control para iniciar o salir del programa.

### prompt generado:

Para generar el programa, implementé un diseño robusto y flexible de cuatro clases: Main, Menus, Operaciones y Utilidades.

La clase Operaciones es el corazón de la lógica de negocio. Definí las constantes para el sueldo base y la comisión del 9%. Su método principal utiliza un ciclo while infinito para solicitar las ventas brutas de manera continua, con la lógica de interrupción (instrucción break) para salir del ciclo cuando se detecta el valor centinela .

La clase Utilidades proporciona soporte multilingüe al centralizar todos los mensajes y textos de la aplicación en dos diccionarios (ES y EN). Por su parte, la clase Menus maneja la interfaz de navegación, presentando el menú de selección de idioma y luego el menú de control (Ejecutar/Finalizar).

La clase Main orquesta el flujo, inicializando el sistema, forzando la selección de idioma y luego entrando en un ciclo de ejecución principal que llama a las Operaciones o finaliza el programa según la opción del usuario.

Salida:

## EJERCICIO#21

### Enunciado

Nomina simple: Desarrolle un programa en JAVA que utilice una instrucción while para determinar el sueldo bruto para cada uno de varios empleados. La empresa paga la cuota normal en las primeras 40 horas de trabajo de cada empleado, y paga cuota y media en todas las horas trabajadas que excedan de 40. Usted recibe una lista de los empleados de la empresa, el número de horas que trabajó cada empleado la semana pasada y la tarifa por horas de cada empleado. Su programa debe recibir como entrada esta información para cada empleado, debe determinar y mostrar el sueldo bruto de cada empleado(Ejercicio 4.16 Libro 1).

### Análisis de Requerimientos

- **Entrada:**
  - Número de empleados (3-5).
  - Por cada empleado: nombre (texto), horas trabajadas ( $\geq 0$ ), tarifa por hora (valida:  $< 80$  si  $\leq 40$  horas,  $> 100$  si  $> 40$  horas).
  - Validación con reintentos si los datos no cumplen.
- **Procesamiento:**
  - Usar un while loop para procesar cada empleado.
  - Calcular sueldo bruto como horas \* tarifa.
  - Validar tarifas según horas trabajadas.
- **Salida:**
  - Mostrar tabla con nombre, horas, tarifa y sueldo.
  - Cuadros elegantes (ancho 70, bordes Unicode).
- **Modularidad:**
  - Ejercicio021\_Main.java: Controla flujo, menús y ciclos.
  - Ejercicio021\_Menu.java: Maneja selección de idioma y menú principal, textos bilingües.
  - Ejercicio021\_Operaciones.java: Calcula sueldo bruto.
  - Ejercicio021\_Utilidades.java: Gestiona formato (cuadros, tabla), entrada de datos, validaciones.
- **Constantes:** ANCHO = 70, HORAS\_NORMALES = 40.0.
- **Estética:** Cuadros simétricos con bordes Unicode, texto centrado.

- **Paradigma:** Programación estructurada, métodos estáticos, Scanner.
- **Documentación:** Sello corto en cada archivo, prompt exacto en Main.

## PROMPT

Agente ahora el ejercicio #21: Nomina simple: Desarrolle un programa en JAVA que utilice una instrucción while para determinar el sueldo bruto para cada uno de varios empleados (Máximo 5 empleados y mínimo 3), La empresa paga la cuota normal en las primeras 40 horas de trabajo de cada empleado, y paga cuota y media en todas las horas trabajadas que excedan de 40. Usted recibe una lista de los empleados de la empresa, el número de horas que trabajó cada empleado la semana pasada y la tarifa por horas de cada empleado, que el número de pago de cuota que sea de 2,000 lempiras y muestra ese dato, Su programa debe recibir como entrada esta información para cada empleado, debe determinar y mostrar el sueldo bruto de cada empleado. Muestra una tabla con los datos ingresados por el usuario, que tenga que ingresar la cantidad de horas trabajadas, si pasa mas de 40 horas paga más "y paga cuota y media en todas las horas trabajadas que excedan de 40". Usa nuestra nueva distribución de archivos:

ejercicio0XX/

```
|—— Ejercicio0XX_Main.java (Flujo principal con ciclos)  
|—— Ejercicio0XX_Menus.java (Menús de idioma y principal)  
|—— Ejercicio0XX_Operaciones.java (Lógica del ejercicio)  
└—— Ejercicio0XX_Utilidades.java (Diseño y formato)
```

. Las nuevas funciones anteriormente mencionadas que solo te recuerdo: "contar con un menú de selección de idioma y luego con un segundo menú donde pueda elegirse entre ejecutarse o interrumpir la ejecución (finalizar el programa).

Corrección: Sueldo = horas \* tarifa (sin cuota base de 2000). Para horas ≤ 40, tarifa < 80; para horas > 40, tarifa > 100, con mensajes de error si no se cumple.

## PROMPT GENERADO:

```

/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 *
 * Jonatan Varela | jivarela@unah.hn | 021 | 2025-10-11
 */
/*
 * PROMPT USADO:
 * Agente ahora el ejercicio #21: Nomina simple: Desarrolla un programa en JAVA que utilice una instrucción while para determinar el sueldo bruto para cada uno de varios empleados (Máximo 5 empleados y mínimo 3). La empresa paga la cuota normal en las primeras 40 horas de trabajo de cada empleado, y paga cuota y media en todas las horas trabajadas que excedan de 40. Usted recibe una lista de los empleados de la empresa, el número de horas que trabajó cada empleado la semana pasada y la tarifa por horas de cada empleado, que el numero de pago de cuota que sea de 2,000 lempiras y muestres ese dato. Su programa debe recibir como entrada esta información para cada empleado, debe determinar y mostrar el sueldo bruto de cada empleado. Muestras una tabla con los datos ingresados por el usuario, que tenga que ingresar la cantidad y horas trabajadas, si pasa mas de 40 horas paga mas "y" paga cuota y media en todas las horas trabajadas que excedan de 40". Usa nuestra nueva distribución de archivos: ejercicio0XX/ —> Ejercicio0XX_Main.java (Flujo principal con ciclos) —> Ejercicio0XX_Menu.java (Menú de idioma y principal) —> Ejercicio0XX_Opciones.java (Lógica del ejercicio) —> Ejercicio0XX_Utilidades.java (Diseño y formato). Las nuevas funciones anteriormente mencionada que solo te recuerdo: "contar con un menú de selección de idioma y luego con un segundo menú donde pueda elegirse entre ejecutarse o interrumpir la ejecución (finalizar el programa).
*/
package Ejercicio021;

import java.util.Scanner;

public class Ejercicio021_Main {
    private static final String LINEA = "-";
    private static final int ANCHO = 70;
    private static final String idiomaActual = "ES";
    ...
}

```

## Salida de ejecución del programa:

```

ERROR: Tariff must be > 100 for > 40 hours
Hourly rate: 120
Hourly rate: 120.0

EMPLOYEE DATA 3

Name: Tromp
Name: Tromp
Hours worked: 40
Hours worked: 40.0
Hourly rate: 70
Hourly rate: 70.0

PAYROLL TABLE

Name | Hours | Rate | Salary
John | 40.0 | 60.00 | 2400.00
Elon | 50.0 | 120.00 | 6000.00
Tromp | 40.0 | 70.00 | 2800.00

Do you want to run again? (Y/N)

```

The screenshot shows the execution of the Java program. It first displays an error message: "ERROR: Tariff must be > 100 for > 40 hours". Then it shows two valid employee entries: "Hourly rate: 120" and "Hourly rate: 120.0". Following this, it prints "EMPLOYEE DATA 3" and lists three employees with their names, hours worked, and hourly rates. Finally, it prints a "PAYROLL TABLE" with the same data, showing the calculated salaries. At the bottom, it asks the user if they want to run again, with options "Y" or "N".

## EJERCICIO#22

### Enunciado

Escriba un programa en pseudocódigo y después una aplicación en JAVA que utilice una instrucción while para determinar e imprimir el mayor número de una serie de 10 números introducidos por el usuario. Su programa debe utilizar tres variables, como se muestra a continuación (Ejercicio 4.17 Libro 1):

- a. contador: Un contador para contar hasta 10 (es decir, para llevar el registro de cuántos números se han introducido, y para determinar cuando se hayan procesado los 10 números).
  - b. número: El número actual que se introduce al programa.
  - c. mayor: El número más grande encontrado hasta ahora
- .

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 022 | 2025-11-04) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Objetivo general Leer 10 números ingresados por el usuario y determinar cuál es el mayor utilizando un ciclo while.
  - Entradas del programa 10 valores numéricos ingresados por el usuario (tipo entero).
  - Salidas del programa Número mayor detectado durante las iteraciones.
  - Restricciones Solo se deben utilizar: contador, número, mayor como variables principales.
  - Extras necesarios (según estándar del proyecto) Selección de idioma, menú para iniciar/salir, sello corto al finalizar.
- **Modularidad:** Cuatro archivos en paquete Ejercicio022:
  - Ejercicio022\_Main.java → Control del flujo principal (incluye prompt en metadatos)
  - Ejercicio022\_Menu.java → Menú de selección de idioma / iniciar / salir
  - Ejercicio022\_Utilidades.java → Estética y sellos gráficos (Unicode)
  - Ejercicio022\_Operaciones.java → Lógica del programa (uso de while para hallar el mayor).

- **Paquete:** Ejercicio022.
- **Estética:** El programa utiliza **bordes elegantes Unicode** para mantener uniformidad visual en toda la solución, haciendo uso del archivo [Ejercicio022\\_Utilidades.java](#).
- **Paradigma:** Programación estructurada (métodos estáticos, sin POO).
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio022\_Menus):**

- seleccionarIdioma() Permite escoger el idioma (ES / EN).
- menuInicioSalir() Pregunta si el usuario desea iniciar o salir del programa.
- obtenerIdioma() Permite a otros módulos saber el idioma seleccionado.

#### **Subrutinas (en Ejercicio022\_Utilidades):**

- mostrarEncabezado(titulo) Imprime el título envuelto en un borde Unicode.
- mostrarPie(mensaje) Muestra un pie de cierre con borde Unicode.
- mostrarSelloCorto() Imprime el sello corto del desarrollador.

#### **Subrutinas (en Ejercicio022\_Operaciones):**

- encontrarMayor()
- Utiliza contador, numero y mayor para determinar el número mayor entre 10 ingresados. Al finalizar, imprime el sello corto y el pie visual.

#### **Flujo Principal (en Ejercicio022\_Main):**

- main()
- Ejecuta selección de idioma → ejecuta menú de inicio → si el usuario continúa → llama a Operaciones.

## **PROMPT**

Agente vamos a desarrollar el ejercicio #22 tomando en cuenta un nuevo menú que nos pide elegir el idioma y otra que nos pide iniciar el programa o salir, estos menús irían en un cuarto archivo llamado Menus.java ejercicio0XX/

```
└── Ejercicio0XX_Main.java (Flujo principal con ciclos)
└── Ejercicio0XX_Menus.java (Menús de idioma y principal)
└── Ejercicio0XX_Operaciones.java (Lógica del ejercicio)
└── Ejercicio0XX_Utilidades.java (Diseño y formato)
```

## PROMPT GENERADO

Perfecto ✅

Aquí tienes la **versión final del Ejercicio #22** con la distribución de 4 archivos, sello personalizado (largo en el Main y corto al final del proceso), estética uniforme con los bordes Unicode, y flujo correcto del menú de idioma + menú de iniciar/salir.

---

 **Estructura Final**

css Copiar código

```
└─ Ejercicio022/
    ├─ Ejercicio022_Main.java      → Control del programa (incluye prompt completo en metadatos)
    ├─ Ejercicio022_Menus.java     → Menú de idioma y menú iniciar/salir
    ├─ Ejercicio022_Utilidades.java → Sellos, cuadros y estética
    └─ Ejercicio022_Operaciones.java → Lógica del ejercicio (WHILE y mayor de 10 números)
```

↓

+ Pregunta lo que quieras 0 

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run Ejercicio022_Main ×
G 1. Iniciar programa
  2. Salir
  Seleccione una opción: 1
MÁXIMO DE 10 NÚMEROS
Ingrese el número #1: 1
Ingrese el número #2: 2
Ingrese el número #3: 3
Ingrese el número #4: 4
Ingrese el número #5: 5
Ingrese el número #6: 6
Ingrese el número #7: 7
Ingrese el número #8: 8
Ingrese el número #9: 9
Ingrese el número #10: 0
-----
El número mayor es: 9
-----
[Dany Noguera | dany.noguera@unah.hn | Ejercicio022]
Proceso finalizado con éxito.
```

## EJERCICIO#23

### Enunciado

¿Cómo te quedó el ojo?: Escriba un programa que pida al usuario que introduzca el tamaño del lado de un cuadrado y que muestre un cuadrado hueco de ese tamaño, compuesto de asteriscos y espacios en blanco. Su programa debe funcionar con cuadrados que tengan lados de todas las longitudes entre 1 y 20. Por ejemplo, si su programa lee un tamaño de 5, debe imprimir: (Ejercicio 4.25 Libro 1).

```
*****  
*   *  
*   *  
*   *  
*****
```

### Análisis

#### Sello Personalizado y Documentación

El Sello Personalizado es la VERSIÓN CORTA: Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 023 | Fecha: 2025-11-03. Se incluye como comentario de bloque en la cabecera de los tres archivos y se muestra en un cuadro estético al final de la ejecución. El prompt completo está en Ejercicio023\_Main.java.

#### Programa y Modularidad

El Programa debe solicitar el tamaño del lado de un cuadrado, el cual debe estar entre 1 y 20. Debe dibujar un cuadrado hueco utilizando asteriscos (\*) y espacios en blanco. La impresión de los asteriscos debe hacerse mediante una sola instrucción de salida (System.out.print('\*')).

La Modularidad usa tres archivos en el paquete Ejercicio023:

1. Ejercicio023\_Main.java: Flujo de control principal, maneja la entrada/salida general.
2. Ejercicio023\_Utilidades.java: Formato estético, cuadros, sello y funciones de entrada (solicitarLado) con validación de rango (1 a 20).
3. Ejercicio023\_Operaciones.java: Contiene la lógica de dibujo (dibujarCuadradoHueco) utilizando ciclos for anidados para determinar si imprimir \* (borde) o un espacio (interior).

El Paradigma es Estructurado, y la Estética usa 70 caracteres de ancho y bordes Unicode.

#### Constantes:

- ANCHO = 70, BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '—'

#### **Subrutinas Clave (Funcionalidad):**

- Ejercicio023\_Operaciones: dibujarCuadradoHueco (implementa la lógica de bucles anidados para la figura).
- Ejercicio023\_Utilidades: solicitarLado (valida el rango de 1 a 20), mostrarSaludo, mostrarEncabezadoResultado, mostrarSelloCorto.

#### **Flujo Principal (en Ejercicio023\_Main.java):**

1. Inicia Scanner. Muestra saludo.
2. Solicita y valida el tamaño del lado (usando Utilidades).
3. Muestra el encabezado del resultado.
4. Llama a Ejercicio023\_Operaciones.dibujarCuadradoHueco para imprimir la figura directamente en la consola.
5. Muestra el pie del resultado y el sello. Cierra Scanner.

#### **PROMPT**

"Agente el #18 hagamos, te recuerdo nuestro sistema de archivo, en este usa los 3 que son: |— Ejercicio018\_Main.java (Programa principal, incluye este prompt) |— Ejercicio018\_Utilidades.java (Funciones de utilidad) |— Ejercicio018\_Operaciones.java (Lógica del ejercicio) Incluye también nuestro sello personalizado.

Necesito que realicemos el ejercicio: ¿Cómo te quedó el ojo?: Escriba un programa que pida al usuario que introduzca el tamaño del lado de un cuadrado y que muestre un cuadrado hueco de ese tamaño, compuesto de asteriscos y espacios en blanco. Su programa debe funcionar con cuadrados que tengan lados de todas las longitudes entre 1 y 20. Por ejemplo, si su programa lee un tamaño de 5, debe imprimir la figura (Ejercicio 4.25 Libro 1)."

#### **Salida del programa**

```

    ;BIENVENIDO AL EJERCICIO #23!
    DIBUJO DE CUADRADO HUECO

    Ingrese el tamaño del lado (entre 1 y 20).

    INGRESO DE TAMAÑO DEL LADO

    Tamaño del lado (1-20):
    >> Ingrese valor: 1

    ** CUADRADO HUECO DE LADO: 1 **

    *

    SELLO PERSONALIZADO

    Kevin Lainez | klainezs@unah.hn | 023 | 2025-11-03

    --- Programa finalizado con éxito ---

    Proceso terminado con código de salida 0
  
```

## EJERCICIO#24

### Enunciado

¿A poco hacen uno? Escriba un programa que lea tres valores double distintos de cero, y que determine e imprima si podrían representar los lados de un triángulo.

### Análisis

El código del Ejercicio 024 (Lados de un Triángulo) confirma su exitosa implementación bajo la Programación Estructurada (PE) y el estricto cumplimiento de los nuevos requisitos de control de flujo e internacionalización. La clase Ejercicio024\_Utilidades actúa como el pilar de la interacción, integrando la lógica bilingüe a través del array textos para los menús de idioma y ejecución, así como la validación robusta (leerDoublePositivo) que garantiza que las entradas sean números double válidos y mayores que cero, manejando errores de tipo (InputMismatchException). Por otro lado, la clase Ejercicio024\_Operaciones mantiene su rol de cálculo puro, verificando la condición de existencia del triángulo; finalmente, Ejercicio024\_Main coordina el flujo, iniciando con la selección de idioma, controlando la repetición del programa mediante un bucle while gestionado por el menú de ejecución, y orquestando la secuencia de lectura, procesamiento y presentación en el idioma seleccionado.

### PROMPT

Creación un código Ejercicio 024 (Lados de un Triángulo) un programa en Java dentro del paquete Ejercicio024 que lea tres valores double positivos, determine si pueden representar los lados de un triángulo y muestre el resultado. La aplicación debe comenzar con un menú de selección de idioma y luego un menú de ejecución (Ejecutar/Finalizar) que controle el flujo del programa. La estructura debe ser de Programación Estructurada (PE), dividida en Ejercicio024\_Operaciones (para implementar la lógica de existencia del triángulo), Ejercicio024\_Utilidades (para gestionar los menús multilingües, el sello, la entrada validada (double > 0) y el formato de la salida), y Ejercicio024\_Main (coordinando la secuencia de I/O y procesamiento dentro de un bucle controlado por el menú de ejecución). El sello (Cristina Cruz... 2025-10-15) debe mantenerse en la documentación de los tres archivos.

### PROMPT GENERADO

The screenshot shows the IntelliJ IDEA interface. On the left, the 'Archivos de Código' (Code Files) sidebar lists three files: 'Ejercicio024\_Main.java', 'Ejercicio024\_Utilidades.java', and 'Ejercicio024\_Operaciones.java'. 'Ejercicio024\_Main.java' is currently selected. The main window displays the Java code for 'Ejercicio024\_Main.java'.

```

package Ejercicio024;
import java.util.Scanner;

/**
 * Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 024 | 2025-10-15
 */
public class Ejercicio024_Main {

    private static final int MSG_LADO_A = 11;
    private static final int MSG_LADO_B = 12;
    private static final int MSG_LADO_C = 13;
    private static final int MSG_FINALIZACION = 18;

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

        Ejercicio024_Utilidades.mostrarSelloCorto();
        Ejercicio024_Utilidades.seleccionarIdioma(scanner);
        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio024_Utilidades.mostrarMenuEjecucion(scanner)) {
                ejecutar = false;
                break;
            }
            Ejercicio024_Utilidades.mostrarEncabezado();
            double ladoA = Ejercicio024_Utilidades.leerDoublePositivo(scanner, Ejercicio024_Utilidades.obtenerTexto(MSG_LADO_A));
            double ladoB = Ejercicio024_Utilidades.leerDoublePositivo(scanner, Ejercicio024_Utilidades.obtenerTexto(MSG_LADO_B));
            double ladoC = Ejercicio024_Utilidades.leerDoublePositivo(scanner, Ejercicio024_Utilidades.obtenerTexto(MSG_LADO_C));
        }
    }
}

```

## Salida de la ejecución del programa:

The screenshot shows the IntelliJ IDEA terminal window titled 'Run' with the tab 'Ejercicio024\_Main' selected. The terminal output shows the execution of the Java program. It starts with the welcome message and asks for language selection. It then presents a menu with options to execute or exit. Finally, it performs a triangle side verification and concludes that the sides can form a triangle.

```

C:\Users\ATA\.jdks\openjdk-25\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2.3\lib\idea_rt.jar=6396,C:\Program Files\JetBrains\IntelliJ IDEA 2025.2.3\lib\ext\javafx.jar=6396" -Dfile.encoding=UTF-8 Ejercicio024_Main
Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 024 | 2025-10-15
*** BIENVENIDO ***
Seleccione el idioma:
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>1

--- Verificación de Lados de un Triángulo ---
Ingrese el lado A (double, positivo): 5
Ingrese el lado B (double, positivo): 4
Ingrese el lado C (double, positivo): 5

--- ANÁLISIS ---
Resultado: Los lados SÍ pueden formar un triángulo.

---- MENÚ DE EJECUCIÓN ---

```

## EJERCICIO #25

### **enunciado:**

¿A poco hacen uno v2?:Escriba un programa que lea tres enteros distintos de cero, y que determine e imprima si podrían ser los lados de un triángulo recto (Ejercicio 4.33 Libro 1).

### **análisis:**

El programa tiene la función de determinar si tres enteros positivos ingresados por el usuario pueden constituir los lados de un triángulo rectángulo. El requisito funcional se resuelve aplicando el Teorema de Pitágoras: el cuadrado del lado más largo (hipotenusa, ) debe ser igual a la suma de los cuadrados de los otros dos lados (). La entrada consiste en tres enteros mayores a cero. El programa debe gestionar errores de entrada y validar que los números sean distintos de cero. La lógica requiere ordenar los lados para identificar la hipotenusa antes del cálculo. El diseño es modular (Main, Operaciones, Menus, Utilidades) e incluye soporte para menús e idiomas. La complejidad del cálculo es , siendo la ordenación o si .

### **prompt**

Crea un programa en Java que solicite al usuario tres enteros positivos que representen los lados de un triángulo. Usa el Teorema de Pitágoras () para verificar si los lados forman un triángulo rectángulo. La solución debe ser modular, capaz de manejar la selección de idioma (español/inglés) y utilizar las clases Operaciones y Menus.

### **prompt generado:**

Para la generación de este programa, implementé la solución utilizando una arquitectura de cuatro clases.

La clase Operaciones contiene la lógica de verificación. Su método principal se encarga de:

1.

Recibir los tres lados como enteros.

2.

Ordenar los valores para identificar la hipotenusa (el lado mayor).

3.

Aplicar el Teorema de Pitágoras: verifica si la suma del cuadrado de los dos lados menores es igual al cuadrado del lado mayor, utilizando tipos long para el cálculo y prevenir desbordamiento.

4.

Devolver un valor booleano o un mensaje que indique el resultado.

Las clases Menus y Utilidades proporcionan una interfaz de usuario completa, incluyendo la selección de idioma y la gestión de mensajes, asegurando que las entradas se soliciten y los resultados se muestren claramente según el idioma elegido. Finalmente, la clase Main coordina la secuencia de ejecución: inicialización, selección de idioma y llamada al método de verificación en Operaciones.

## Salida:

```
Windows PowerShell      x + v - □ ×
Advertencia: PowerShell detectó que es posible que estés usando un lector de pantalla y que hayas deshabilitado PSReadLine con fines de compatibilidad. Siquieres volver a habilitarlo, ejecuta "Import-Module PSReadLine".
PS C:\Users\erick> cd "C:\Users\erick\Documents\Proyectos Java\Guía intermedia"
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> javac -d out src\Ejercicio_025/*.java
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> java -cp out Ejercicio_025.Main
--- SELECCIÓN DE IDIOMA / LANGUAGE SELECTION ---
1. Español
2. Inglés (English)
Seleccione el idioma (1 o 2): 1

--- MENÚ DE CONTROL ---
1. Ejecutar Verificación de Triángulo Rectángulo
2. Finalizar el Programa
Seleccione una opción (1 o 2): 1

--- MÓDULO: TRIÁNGULO RECTÁNGULO (TEOREMA DE PITÁGORAS) ---
Introduzca las longitudes (enteros positivos) de los tres lados del triángulo:
Lado 1: 5
Lado 2: 12
Lado 3: 13

RESULTADO: Los lados 5, 12 y 13 PUEDEN formar un triángulo rectángulo ( $a^2 + b^2 = c^2$ ).
--- MENÚ DE CONTROL ---
1. Ejecutar Verificación de Triángulo Rectángulo
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

¡Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## EJERCICIO#26

### Enunciado:

Copiar y pegar: Escriba una aplicación en JAVA que utilice una instrucción while y la secuencia de escape de tabulación \t para imprimir la siguiente tabla de valores: (Ejercicio 4.18 Libro 1).

N	10*N	100*N	1000*N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000

### El programa debe:

- Mostrar un sello personalizado con salida elegante y formal.
- Incluir un menú de selección de idioma (Español/English) y un menú principal (Ejecutar/Salir).
- Permitir repetir la ejecución tras cada ciclo.
- Usar la distribución de cuatro archivos: Main.java (flujo principal), Menus.java (menús), Operaciones.java (lógica), Utilidades.java (formato).

### Análisis de Requerimientos

- **Entrada:** Ninguna (tabla generada automáticamente).
- **Procesamiento:**
  - Usar un while loop para iterar de 1 a 5.
  - Calcular y mostrar N, 10\*N, 100\*N, 1000\*N usando tabulaciones (\t).
- **Salida:**
  - Imprimir tabla en un cuadro elegante (ancho 70, bordes Unicode).
  - Incluir encabezado y filas alineadas con tabulaciones.
- **Modularidad:**
  - Main.java: Controla flujo, menús y ciclos.
  - Menus.java: Maneja selección de idioma y menú principal, textos bilingües.
  - Operaciones.java: Genera la tabla de valores.
  - Utilidades.java: Gestiona formato (cuadros, tabulación), salida de datos.
- **Constantes:** ANCHO = 70.
- **Estética:** Cuadros simétricos con bordes Unicode, texto centrado.
- **Paradigma:** Programación estructurada, métodos estáticos.
- **Documentación:** Sello corto en cada archivo, prompt exacto en Main.

## PROMPT

Agente un nuevo ejercicio el #26, recuerda los parámetros antes desarrollados en ejercicios anteriores, el ejercicio conta de los siguiente:

Copiar y pegar: Escriba una aplicación en JAVA que utilice una instrucción while y la secuencia de escape de tabulación \t para imprimir la siguiente tabla de valores:

N	10*N	100*N	1000*N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000

Implementa el diseño de 4 archivos, con un cuadro elegante con la salida, junto al menú que corresponde a los ejercicios que hemos desarrollado.

## PROMPT GENRADO

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 026 | 2025-10-12
 */
package Ejercicio026;

public class Utilidades {
    private static final int ANCHO = 70;
    private static final char BORDE_IQZ = '|';
    private static final char BORDE_DER = '|';
    private static final char LINEA_H = '-';

    public static void mostrarSello() {
        System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "=");
        System.out.print(BORDE_IQZ + centrarTexto("SELLO PERSONALIZADO - VERSIÓN CORTA", ANCHO - 2) + BORDE_DER);
        System.out.print(BORDE_IQZ + centrarTexto("Jonatan Varela | jivarelag@unah.hn | 026 | 2025-10-12", ANCHO - 2) + BORDE_DER);
        System.out.println("=" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "=");
    }

    public static void mostrarTablaValores(String idioma) {
        System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "=");
        System.out.print(BORDE_IQZ + centrarTexto(obtenerTexto("TITULO_TABLA", idioma), ANCHO - 2) + BORDE_DER);
        System.out.print("=" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "=");
        System.out.print(BORDE_IQZ + centrarTexto("\n\t10*N\t100*N\t1000*N", ANCHO - 2) + BORDE_DER);
        System.out.println("=" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "=");
    }
}
```

## Salida de la ejecucion del programa:

```
SELECCIÓN DE IDIOMA / LANGUAGE SELECTION
1. Español
2. English

Seleccione su idioma / Select language: 1
|Idioma seleccionado / Selected language: Español

MENÚ PRINCIPAL
1. Ejecutar programa
2. Salir

Seleccione una opción: 1
|Seleccione una opción: 1

TABLA DE VALORES
N 10*N 100*N 1000*N
1 10 100 1000
```



## EJERCICIO#27

### Enunciado

Escriba un programa que utilice una instrucción for para sumar una secuencia de enteros. Suponga que el primer entero leído especifica el número de valores que quedan por introducir. Su programa debe leer sólo un valor por cada instrucción de entrada. Una secuencia típica de entrada podría ser:

5 100 200 300 400 500

donde el 5 indica que se van a sumar los 5 valores subsiguientes.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 027 | 2025-11-04) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
  - **Programa:** Objetivo general Sumar una secuencia de enteros ingresados por el usuario usando un bucle for.
    - Entradas del programa - Primer entero: número de valores a sumar.
    - Valores enteros subsiguientes: los números que se sumarán.
    - Salidas del programa: - Suma total de los valores ingresados.
    - Restricciones: Solo se lee un valor por instrucción de entrada. El primer número indica cuántos números serán leídos después.
    - Extras necesarios: Menú de selección de idioma, menú de iniciar/salir, sello corto al finalizar.
  - **Modularidad:** Cuatro archivos en paquete Ejercicio027:
    - Ejercicio027\_Main.java → Control del flujo principal (incluye prompt en metadatos)
    - Ejercicio027\_Menus.java → Menú de idioma y menú iniciar/salir
    - Ejercicio027\_Utilidades.java → Sellos, cuadros y formato estético
    - Ejercicio027\_Operaciones.java → Lógica de sumatoria usando for
  - **Paquete:** Ejercicio027.
  - **Estética:** Bordes Unicode para todos los títulos, secciones y pies de página. Encabezado al iniciar cada módulo.
- Sello corto antes del mensaje de cierre.  
Pie final con mensaje de operación completada.
- **Paradigma:** Programación estructurada y modular, con separación de responsabilidades.

Flujo controlado desde Main.  
Presentación y estética separadas en Utilidades.  
Lógica matemática en Operaciones.  
Interacción con usuario (menus) en Menus.

- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio027\_Menu):**

- seleccionarIdioma() Permite escoger el idioma (ES / EN).
- menuInicioSalir() Permite decidir si iniciar el programa o salir.
- obtenerIdioma() Permite a otros módulos obtener el idioma seleccionado.

#### **Subrutinas (en Ejercicio027\_Utilidades):**

- mostrarEncabezado(String titulo) Imprime un encabezado con borde Unicode.
- mostrarPie(String mensaje) Imprime pie de pantalla con borde Unicode.
- mostrarSelloCorto() Imprime el sello corto del desarrollador antes del pie final.

#### **Subrutinas (en Ejercicio027\_Operaciones):**

- sumarSecuencia()
- Solicita el primer número (cantidad de valores), luego recorre la secuencia con for sumando cada valor. Al finalizar, imprime el sello corto y el pie final.

#### **Flujo Principal (en Ejercicio027\_Main):**

- main()
- Ejecuta la selección de idioma → menú de inicio → llama a Operaciones.sumarSecuencia() si el usuario inicia ejecución.

#### **PROMPT**

Agente vamos con el Ejercicio #27, exactamente quiero todo lo que hicimos en el ejercicio 22 en cuanto estructura y todo lo que te he especificado, el enunciado dice: Escriba un programa que utilice una instrucción for para sumar una secuencia de enteros. Suponga que el primer entero leído especifica el número de valores que quedan por introducir. Su programa debe leer sólo un valor por cada instrucción de entrada. Una secuencia típica de entrada podría ser:

5 100 200 300 400 500

donde el 5 indica que se van a sumar los 5 valores subsiguientes

## PROMPT GENERADO

Perfecto ✓

Vamos a generar el Ejercicio #27 siguiendo exactamente la misma estructura y estándares que aplicamos en el Ejercicio 22:

- Distribución en 4 archivos: Main, Menus, Utilidades, Operaciones .
- Sello largo en Main con prompt completo.
- Menús: selección de idioma y opción iniciar/salir.
- Estética uniforme con bordes Unicode.
- Sello corto al finalizar operaciones.

### Estructura del Ejercicio #27

markdown

Copiar código

```
└─ Ejercicio027/
    ├─ Ejercicio027_Main.java
    ├─ Ejercicio027_Menus.java
    ├─ Ejercicio027_Utilidades.java
    └─ Ejercicio027_Operaciones.java
```

+ | Pregunta lo que quieras

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run Ejercicio027_Main ×
G 1. Español
  2. English
  Seleccione /_Select: 1
  MENÚ PRINCIPAL
  1. Iniciar programa
  2. Salir
  Seleccione una opción: 1
  SUMA DE SECUENCIA DE ENTEROS
  Ingrese la cantidad de valores a sumar: 4
  Ingrese el valor #1: 1
  Ingrese el valor #2: 5
  Ingrese el valor #3: 6
  Ingrese el valor #4: 8
  -----
  La suma total es: 20
  -----
  [Dany Noguera | Ejercicio027]
  Proceso finalizado con éxito.
```

## EJERCICIO#28

### Enunciado

Criptografía: Una compañía desea transmitir datos a través del teléfono, pero le preocupa que sus teléfonos puedan estar intervenidos. Todos los datos se transmiten como enteros de cuatro dígitos. La compañía le ha pedido a usted que escriba un programa que cifre sus datos, de manera que éstos puedan transmitirse con más seguridad. Su programa debe leer un entero de cuatro dígitos introducido por el usuario y cifrarlo de la siguiente manera: reemplace cada dígito con (el resultado de sumar 7 al dígito) módulo 10. Luego intercambie el primer dígito con el tercero, e intercambie el segundo dígito con el cuarto. Después imprima el entero cifrado. Escriba un programa separado que reciba como entrada un entero de cuatro dígitos cifrado, y que lo descifre para formar el número original

### Análisis

El código Java corresponde a la clase principal Ejercicio028\_Main es un programa que implementa un sistema de cifrado y descifrado simple, probablemente de números de cuatro dígitos, con soporte para múltiples idiomas. La estructura del programa es altamente modular, delegando la funcionalidad a otras clases como Ejercicio028\_Utilidades (para interacción con el usuario, selección de idioma, menús, lectura de datos y visualización de resultados) y Ejercicio028\_Operaciones (para la lógica de cifrado y descifrado). El método main establece un bucle principal (do-while) que permite al usuario seleccionar un idioma y luego, dentro de un bucle interno (while), acceder a un menú para cifrar (opción 1), descifrar (opción 2), cambiar el idioma (opción 3 en el menú interno), o salir completamente del programa (opción 4 o seleccionando el idioma 3). Se utiliza una única instancia estática de Scanner para la entrada de datos, la cual se cierra limpiamente al finalizar la ejecución, siguiendo buenas prácticas de manejo de recursos.

### PROMPT

Genera el código Java completo para las clases Ejercicio028\_Utilidades y Ejercicio028\_Operaciones basado en el main proporcionado. La clase Ejercicio028\_Operaciones debe implementar un cifrado y descifrado de números de cuatro dígitos (0000-9999) donde el cifrado consiste en sustituir cada dígito \$d\$ por \$(d+7) \% 10\$, y luego intercambiar los dígitos de posición (1ro con 3ro, 2do con 4to), retornando un String de 4 caracteres. La clase Ejercicio028\_Utilidades debe gestionar la interfaz de usuario en Español (1) e Inglés (2), implementando todos los métodos llamados en el main para la selección de idioma, la visualización de menús, la lectura y validación de números de 4 dígitos, y la presentación de resultados y despedida, asegurando que el programa sea completamente funcional y modular." recuerda debe llevar los mismos elemento que los codigos anteriores.

## PROMPT GENERADO

[Ejercicio028\\_Main](#), [Ejercicio028\\_Utilidades](#) y [Ejercicio028\\_Operaciones](#). El programa debe implementar un **sistema de cifrado y descifrado de números enteros de exactamente 4 dígitos** (0000 a 9999) para transmisión segura, siguiendo este algoritmo:

- **Cifrado:**

1. Reemplazar cada dígito  $d$  por  $(d + 7) \% 10$ .
2. Intercambiar el **primer dígito con el tercero**.
3. Intercambiar el **segundo dígito con el cuarto**.
4. Devolver el número cifrado como cadena de 4 dígitos (con ceros a la izquierda).

- **Descifrado:**

1. Intercambiar primero con tercero, segundo con cuarto (inverso del intercambio).
2. Reemplazar cada dígito  $d$  por  $(d + 3) \% 10$  (inverso de  $+7 \bmod 10$ ).
3. Devolver el número original como cadena de 4 dígitos.

El programa debe comenzar mostrando un **sello personalizado centrado** en un marco de 70 caracteres con:

text

Razonamiento más profundo x Contraer Ajuste Cop v

## Salida de la ejecución del programa:

```
-----
===== SELECCIONE EL IDIOMA / SELECT LANGUAGE =====
1. Español
2. English
3. Salir / Exit
Opción / Option: 1

-----
MENÚ PRINCIPAL
-----
1. Cifrar un número
2. Descifrar un número
3. Volver al menú de idioma
4. Salir del programa
Seleccione una opción: 1

Ingrese un número de 4 dígitos para cifrar (0000-9999): (0000-9999): 4874

RESULTADO DEL {tipo.toUpperCase()}
Original: 4874 → Cifrado/Descifrado: 4115
```

## EJERCICIO#29

### Enunciado

Promedio: Escriba un programa que utilice una instrucción for para calcular e imprimir el promedio de varios enteros. Suponga que el último valor leído es el valor centinela 9999. Una secuencia típica de entrada podría ser: 10 8 11 7 9 9999 lo cual indica que el programa debe calcular el promedio de todos los valores antes del 9999

### Análisis

El código del Ejercicio 029 (Promedio con Centinela) confirma su correcta implementación en Programación Estructurada (PE), respetando la modularidad y los nuevos requisitos de control de flujo e internacionalización. La clase Ejercicio029\_Operaciones encapsula puramente la lógica matemática, proporcionando el método calcularPromedio con la debida protección contra la división por cero. La clase Ejercicio029\_Utilidades gestiona toda la interacción con el usuario, incluyendo la internacionalización bilingüe a través del array textos para los menús de idioma y ejecución, la validación de entrada (leerEntero con manejo de InputMismatchException), y el formateo de la salida (mostrarResultado). Finalmente, Ejercicio029\_Main cumple con el requisito específico del ejercicio al utilizar un bucle for con una lógica de centinela especial (9999) para controlar la lectura y acumulación de números, asegurando que el flujo general sea controlado por el menú de ejecución que permite al usuario repetir la operación cuantas veces desee.

### PROMPT

Creame un código del Ejercicio 029 (Promedio con Centinela) un programa en Java dentro del paquete Ejercicio029 que lea una secuencia de números enteros y calcule e imprima su promedio, utilizando el valor 9999 como valor centinela para indicar el final de la entrada de datos. La aplicación debe comenzar con un menú de selección de idioma y luego un menú de ejecución (Ejecutar/Finalizar) que controle el flujo del programa. La estructura debe ser de Programación Estructurada (PE), dividida en Ejercicio029\_Operaciones (para calcular el promedio y proteger contra la división por cero), Ejercicio029\_Utilidades (para gestionar los menús multilingües, el sello, la entrada validada de enteros, y el formato de la salida), y Ejercicio029\_Main (coordinando la secuencia de I/O y procesamiento, e implementando la lógica del centinela dentro de un bucle for). El sello (Cristina Cruz... 2025-10-15) debe mantenerse en la documentación de los tres archivos.

## PROMPT GENERADO

The screenshot shows a code editor interface with three Java files listed under 'Archivos de Código': **Ejercicio029\_Main.java**, **Ejercicio029\_Utilidades.java**, and **Ejercicio029\_Operaciones.java**. The code for **Ejercicio029\_Main.java** is displayed on the right:

```
package Ejercicio029;
import java.util.Scanner;

/*
 * Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 029 | 2025-10-15
 */
public class Ejercicio029_Main {

    private static final int MSG_ENTRADA = 11;
    private static final int MSG_FINALIZACION = 15;

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

        Ejercicio029_Utilidades.mostrarSelloCorto();

        Ejercicio029_Utilidades.seleccionarIdioma(scanner);

        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio029_Utilidades.mostrarMenuEjecucion(scanner)) {
                ejecutar = false;
                break;
            }

            Ejercicio029_Utilidades.mostrarEncabezado();

            double suma = 0.0;
            int contador = 0;
            int centinela = Ejercicio029_Utilidades.getCentinela();
        }
    }
}
```

## Salida de la ejecución del programa:

The screenshot shows a terminal window titled 'Run' with the tab 'Ejercicio029\_Main' selected. The output of the program is displayed in the terminal window:

```
Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 029 | 2025-10-15

*** BIENVENIDO ***
Seleccione el idioma:
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>1

--- Cálculo de Promedio con Centinela ---
Ingrese un número entero (9999 para finalizar): 58
Ingrese un número entero (9999 para finalizar): 78
Ingrese un número entero (9999 para finalizar):
45
Ingrese un número entero (9999 para finalizar): 9999
El promedio de los 3 números es: 60.33

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
```

## EJERCICIO #30

enunciado:

El menor de todos tus males: Escriba un programa que utilice una instrucción for para encontrar el menor de varios enteros. Suponga que el primer valor leído especifica el número de valores restantes (Ejercicio 5.8 Libro 1).

análisis:

El objetivo de este programa es encontrar el valor entero más pequeño dentro de una secuencia de números. La entrada se realiza en dos fases: primero, se solicita un entero que indica la cantidad de números a comparar (el primer valor leído); luego, se leen los valores restantes dentro de un ciclo for. La lógica principal inicializa una variable menorValor con el primer número de la secuencia y lo actualiza solo si se encuentra un valor subsiguiente que sea menor. El programa debe manejar entradas inválidas (no enteros o ) y está modularizado en Operaciones (lógica de búsqueda) y clases de soporte para menús e idiomas. La complejidad del algoritmo de búsqueda es lineal.

prompt

Crea un programa en Java que encuentre el menor valor de un conjunto de enteros. Primero, solicita un número entero que indique cuántos valores se leerán a continuación. Luego, utiliza un ciclo for para leer esos números y determinar el más pequeño. El resultado final debe ser solo el valor mínimo. La solución debe ser modular, incluyendo clases para la lógica (Operaciones) y la gestión de menús e idiomas.

prompt generado:

Para la generación de este programa, la lógica se concentró en la clase Operaciones, siguiendo el flujo de trabajo de dos pasos.

Primero, se implementó la solicitud y validación de la cantidad de números a procesar (). Si es , el proceso se cancela.

Luego, se utilizó la instrucción for para iterar exactamente veces. Dentro del ciclo:

1.

Se inicializó una variable llamada menorValor con el primer número que el usuario ingresó de la secuencia.

2.

Para cada iteración subsiguiente, el nuevo número ingresado se comparó directamente con menorValor. Si el nuevo número era estrictamente menor, menorValor se actualizó.

3.

Se implementó una validación de la entrada para asegurar que el usuario siempre ingrese un entero, repitiendo la solicitud para el mismo índice si la entrada era incorrecta.

Finalmente, el Main coordinó la ejecución, y las clases Menus y Utilidades facilitaron la interacción con el usuario en el idioma seleccionado y mostraron el menorValor encontrado.

### Salida:

```
Windows PowerShell x + PS C:\Users\erick> cd "C:\Users\erick\Documents\Proyectos Java\Guía intermedia"
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> javac -d out src\Ejercicio_030/*.java
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> java -cp out Ejercicio_030.Main
--- SELECCIÓN DE IDIOMA / LANGUAGE SELECTION ---
1. Español
2. Inglés (English)
Seleccione el idioma (1 o 2): 1

--- MENÚ DE CONTROL ---
1. Ejecutar Búsqueda del Menor Valor
2. Finalizar el Programa
Seleccione una opción (1 o 2): 1

--- MÓDULO: EL MENOR DE VARIOS ENTEROS (CICLO FOR) ---
Ingrese la cantidad de números que va a comparar (N > 0): 4
Proceda a ingresar los 4 valores restantes.
Valor 1: 12
Valor 2: 7
Valor 3: 25
Valor 4: 3

El menor de los valores ingresados es: 3

--- MENÚ DE CONTROL ---
1. Ejecutar Búsqueda del Menor Valor
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

¡Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## EJERCICIO#31

### Enunciado:

No pases de 15: Escriba un programa que utilice una instrucción for para calcular e imprimir el producto de los enteros impares del 1 al 15(Ejercicio 5.9 Libro 1).

### Análisis de Requerimientos

- **Entrada:** Ninguna (cálculo automático).
- **Procesamiento:**
  - Usar un for loop para iterar sobre impares (1, 3, 5, 7, 9, 11, 13, 15).
  - Calcular el producto paso a paso y almacenar los pasos.
  - Calcular la suma del resultado final (que será igual al producto en este caso).
- **Salida:**
  - Mostrar los pasos matemáticos en un cuadro elegante (ancho 70, bordes Unicode).
  - Incluir el producto final y su suma.
- **Modularidad:**
  - Main.java: Controla flujo, menús y ciclos.
  - Menus.java: Maneja selección de idioma y menú principal, textos bilingües.
  - Operaciones.java: Calcula el producto de impares con pasos detallados.
  - Utilidades.java: Gestiona formato (cuadros, salida), muestra resultados.
- **Constantes:** ANCHO = 70.
- **Estética:** Cuadros simétricos con bordes Unicode, texto centrado.
- **Paradigma:** Programación estructurada, métodos estáticos.
- **Documentación:** Sello corto en cada archivo, prompt exacto en Main.

### PROMPT

Agente el ejercicio#31, siempre muestra nuestro nuevo sistema de distribucion de carpetas, con los menús atractivos en los cuadros anteriormente desarrollados, el nuevo ejercicio consta de los siguiente:

No pases de 15: Escriba un programa que utilice una instrucción for para calcular e imprimir el producto de los enteros impares del 1 al 15.

Todo en una salida elegante al usuario, y ahora usemos un mensaje de despedida cuando el usuario deje de usar el programa.

muestra todos los procesos matemáticos que haces y los muestras como esto:

$$1 \times 3 = 3$$

$$3 \times 5 = 15$$

$$15 \times 7 = 105$$

$$105 \times 9 = 945$$

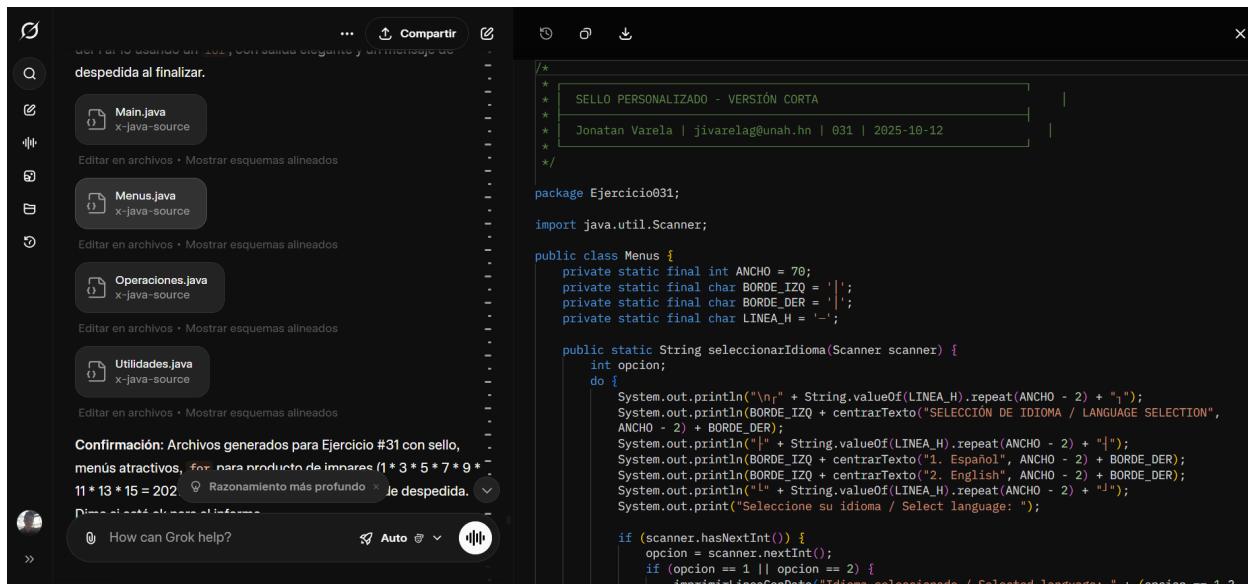
$$945 \times 11 = 10395$$

$$10395 \times 13 = 135135$$

$$135135 \times 15 = 2\,027\,025 \rightarrow \text{producto final} = 2,027,025$$

y muestra la suma que haces del resultado de los productos

## PROMPT GENERADO



```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 031 | 2025-10-12
 */

package Ejercicio031;

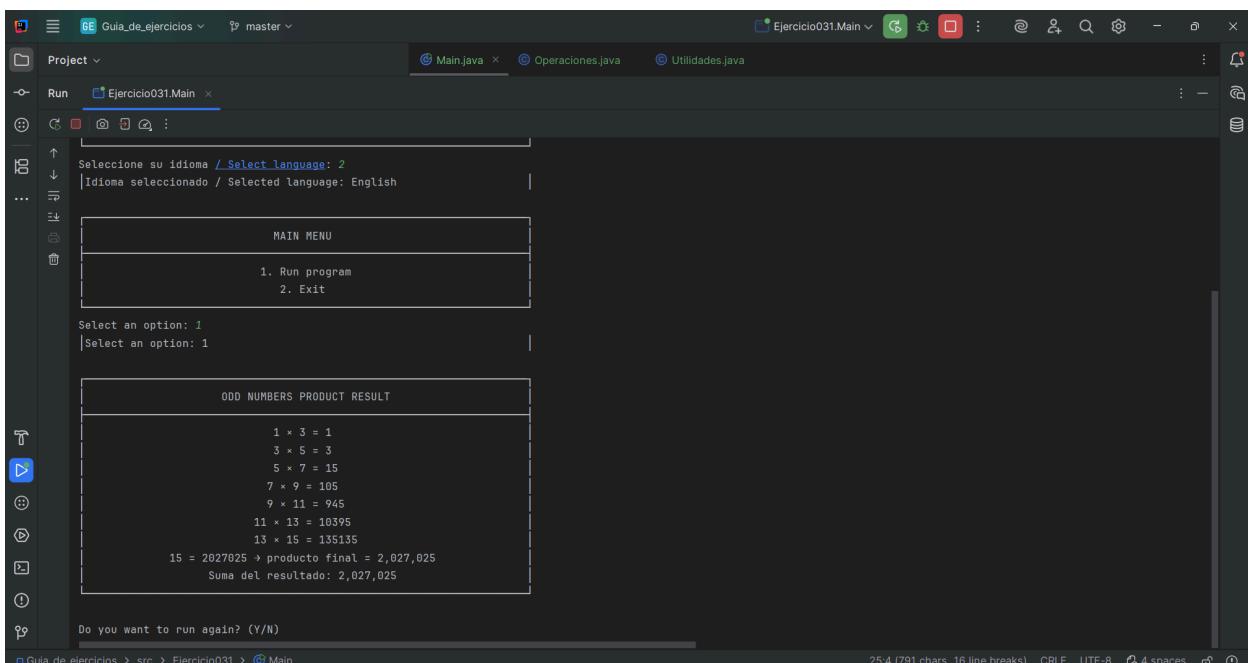
import java.util.Scanner;

public class Menus {
    private static final int ANCHO = 70;
    private static final char BORDE_IZQ = '|';
    private static final char BORDE_DER = '|';
    private static final char LINEA_H = '-';

    public static String seleccionarIdioma(Scanner scanner) {
        int opcion;
        do {
            System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
            System.out.println(BORDE_IZQ + centrarTexto("SELECCIÓN DE IDIOMA / LANGUAGE SELECTION",
                ANCHO - 2) + BORDE_DER);
            System.out.println("|" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
            System.out.println(BORDE_IZQ + centrarTexto("1. Español", ANCHO - 2) + BORDE_DER);
            System.out.println(BORDE_IZQ + centrarTexto("2. English", ANCHO - 2) + BORDE_DER);
            System.out.println(LINEA_H + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
            System.out.print("Seleccione su idioma / Select language: ");

            if (scanner.hasNextInt()) {
                opcion = scanner.nextInt();
                if (opcion == 1 || opcion == 2) {
                    System.out.println("Idioma seleccionado / Selected language: " + opcion == 1 ? "Español" : "English");
                }
            }
        } while (opcion != 1 & opcion != 2);
    }
}
```

## Salida de la ejecución del programa:



```
Seleccione su idioma / Select language: 2
idioma seleccionado / Selected language: English

MAIN MENU
1. Run program
2. Exit

Select an option: 1
Select an option: 1

ODD NUMBERS PRODUCT RESULT
1 × 3 = 1
3 × 5 = 3
5 × 7 = 15
7 × 9 = 105
9 × 11 = 945
11 × 13 = 10395
13 × 15 = 135135
15 = 2027025 → producto final = 2,027,025
Suma del resultado: 2,027,025

Do you want to run again? (Y/N)
```

## EJERCICIO#32

### Enunciado

Función factorial: La función factorial se utiliza frecuentemente en los problemas de probabilidad. Utilizando la definición de factorial, escriba un programa que utilice una función for para evaluar los factoriales de los enteros del 1 al 5. Muestre los resultados en formato tabular.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 032 | 2025-11-04) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Objetivo general      Calcular y mostrar los factoriales de los enteros del 1 al 5 utilizando un bucle for.

Entradas del programa      Ninguna entrada externa requerida para los números a calcular; los números son fijos (1 al 5).

Salidas del programa      Tabla con dos columnas: número y su factorial.

Restricciones Se debe usar un bucle for para calcular cada factorial.

Extras necesarios Menú de selección de idioma, menú de iniciar/finalizar, bucle principal que permita ejecutar múltiples veces.

- **Modularidad:** Cuatro archivos en paquete Ejercicio032:
  - Ejercicio032\_Main.java → Control del flujo principal (prompt incluido)
  - Ejercicio032\_Menus.java → Menú de idioma y menú iniciar/finalizar
  - Ejercicio032\_Utilidades.java → Sellos, cuadros y formato estético
  - Ejercicio032\_Operaciones.java → Lógica de cálculo de factoriales.

- **Paquete:** Ejercicio032.

- **Estética:** Bordes Unicode („, „, ||, =) para encabezados, secciones y pies de página.

Encabezado al iniciar el módulo de operaciones con título centrado.

Sello corto antes del mensaje de cierre, manteniendo uniformidad.

Pie final solo se muestra después de cada ejecución exitosa.

- **Paradigma:** Programación estructurada y modular.

Flujo controlado desde Main.

Presentación y estética separadas en Utilidades.

Lógica matemática en Operaciones.

Interacción con usuario (menus) en Menus.

Bucle principal que permite ejecutar múltiples veces hasta que el usuario decida finalizar.

- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio032\_Menu):**

- seleccionarIdioma() Permite escoger el idioma (ES / EN).
- menuInicioSalir() Permite decidir si iniciar el programa o salir; devuelve boolean para controlar bucle en Main.
- obtenerIdioma() Permite a otros módulos obtener el idioma seleccionado.

#### **Subrutinas (en Ejercicio032\_Utilidades):**

- mostrarEncabezado(String título) Imprime encabezado con borde Unicode.
- mostrarPie(String mensaje) Imprime pie de pantalla con borde Unicode al finalizar cada ejecución.
- mostrarSelloCorto() Imprime el sello corto del desarrollador antes del pie final.

#### **Subrutinas (en Ejercicio032\_Operaciones):**

- calcularFactoriales()
- Recorre los enteros 1 al 5 usando un bucle for, calcula factorial de cada número, imprime tabla con resultados, muestra sello corto y pie final.

#### **Flujo Principal (en Ejercicio032\_Main):**

- main()
- Ejecuta la selección de idioma → bucle principal con menú de inicio/finalizar → llama a `Operaciones.calcularFactoriales()` si el usuario inicia ejecución.

### **PROMPT**

Agente vamos con el Ejercicio #32, exactamente quiero todo lo que hicimos en los ejercicios anteriores en cuanto estructura y todo lo que te he especificado, el enunciado dice: :Función factorial: La función factorial se utiliza frecuentemente en los problemas de probabilidad. Utilizando la definición de factorial, escriba un programa que utilice una función for para evaluar los factoriales de los enteros del 1 al 5. Muestre los resultados en formato tabular.

## PROMPT GENERADO

Perfecto 😊 Vamos a generar el **Ejercicio #32** siguiendo **exactamente la misma estructura y estándares** que aplicamos en el Ejercicio 27:

- Distribución de **4 archivos**: `Main`, `Menus`, `Utilidades`, `Operaciones`.
- Sello largo en `Main` con **prompt completo**.
- Menús: selección de idioma y opción iniciar/salir.
- Estética uniforme con **bordes Unicode**.
- Sello corto antes del mensaje final.

**Estructura del Ejercicio #32**

markdown Copiar código

```
└── Ejercicio032/
    ├── Ejercicio032_Main.java
    ├── Ejercicio032_Menus.java
    ├── Ejercicio032_Utilidades.java
    └── Ejercicio032_Operaciones.java
```

+ Pregunta lo que quieras Q

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run  Ejercicio032_Main  x
[  ]  [  ]  [  ]  [  ]
↑ 1. Español
↓ 2. English
[  ] Seleccione /_Select: 1
[  ] MENÚ PRINCIPAL
[  ] 1. Iniciar programa
[  ] 2. Finalizar
[  ] Seleccione una opción: 1
[  ] FACTORIALES DEL 1 AL 5
[  ] Número Factorial
[  ] -----
[  ] 1      1
[  ] 2      2
[  ] 3      6
[  ] 4     24
[  ] 5    120
[  ] [Dany Noguera | Ejercicio032]
[  ] Proceso finalizado con éxito.
```

## EJERCICIO#33

### Enunciado

Patrones: Escriba un programa que utilice ciclos for para generar los siguientes patrones por separado, uno debajo del otro. Use ciclos for para generar los patrones. Todos los asteriscos (\*) deben imprimirse mediante una sola instrucción de la forma cout << "\*"; (esto hace que los asteriscos se impriman uno al lado del otro). [Sugerencia: los últimos dos patrones requieren que cada línea empiece con un número apropiado de espacios en blanco. Crédito adicional: combine su código de los cuatro problemas separados en un solo programa que imprima los cuatro patrones, uno al lado del otro, haciendo un uso inteligente de los ciclos for anidados.]

### Análisis

#### Sello Personalizado y Documentación

El Sello Personalizado es la VERSIÓN CORTA: Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 033 | Fecha: 2025-11-03. Se incluye como comentario de bloque en la cabecera de los tres archivos y se muestra en un cuadro estético al final de la ejecución. El prompt completo está en Ejercicio033\_Main.java.

#### Programa y Modularidad

El Programa debe generar cuatro patrones de asteriscos (\*) utilizando ciclos for anidados. Los patrones son:

- (a) Triángulo creciente (izquierda).
- (b) Triángulo decreciente (izquierda).
- (c) Triángulo decreciente (derecha).
- (d) Triángulo creciente (derecha).

Los patrones deben imprimirse uno debajo del otro. La impresión de asteriscos y espacios debe hacerse usando una sola instrucción de salida por carácter (cumpliendo el requisito de cout << "\*"). Se asume un tamaño de base de 10.

La Modularidad usa tres archivos en el paquete Ejercicio033:

1. Ejercicio033\_Main.java: Flujo de control principal, llama secuencialmente a las funciones de dibujo.
2. Ejercicio033\_Utilidades.java: Formato estético, cuadros y sello.
3. Ejercicio033\_Operaciones.java: Contiene la lógica de dibujo para los cuatro patrones, manejando la impresión de caracteres y espacios.

El Paradigma es Estructurado, y la Estética usa 70 caracteres de ancho y bordes Unicode.

## Constantes:

- ANCHO = 70, BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '-'
  - TAMANO\_BASE = 10 (Asumido para el dibujo).

### **Subrutinas Clave (Funcionalidad):**

- Ejercicio033\_Operaciones:
    - dibujarPatronA, dibujarPatronB, dibujarPatronC, dibujarPatronD  
(Lógica de bucles anidados).
    - imprimirCaracter (Función auxiliar para imprimir el \* o repetidamente).
  - Ejercicio033\_Utilidades: mostrarSaludo, mostrarEncabezadoPatrones, mostrarPiePatrones, mostrarSelloCorto.

### **Flujo Principal (en Ejercicio033\_Main.java):**

1. Muestra saludo.
  2. Muestra un encabezado para la sección de patrones.
  3. Llama secuencialmente a Ejercicio033\_Operaciones.dibujarPatronA() hasta dibujarPatronD().
  4. Muestra el pie del resultado.
  5. Muestra el sello. Finaliza el programa.

## PROMPT

"Agente el #18 hagamos, te recuerdo nuestro sistema de archivo, en este usa los 3 que son: |— Ejercicio033\_Main.java (Programa principal, incluye este prompt) |— Ejercicio033\_Utilidades.java (Funciones de utilidad) |— Ejercicio033\_Operaciones.java (Lógica del ejercicio) Incluye también nuestro sello personalizado.

Necesito que realicemos el ejercicio: Patrones: Escriba un programa que utilice ciclos for para generar los siguientes patrones por separado, uno debajo del otro. Use ciclos for para generar los patrones. Todos los asteriscos (\*) deben imprimirse mediante una sola instrucción de la forma cout << "(Ejercicio 5.12 Libro 1)."

## **Salida del programa**

## EJERCICIO#34

### Enunciado

Te vendo un pato: Un almacén de pedidos por correo vende cinco productos distintos, cuyos precios de venta son los siguientes: producto 1, \$2.98; producto 2, \$4.50; producto 3, \$9.98; producto 4, \$4.49 y producto 5, \$6.87. Escriba un programa que lea una serie de pares de números, como se muestra a continuación:

- A. número del producto;
- B. cantidad vendida.

Su programa debe utilizar una instrucción switch para determinar el precio de venta de cada producto. Debe calcular y mostrar el valor total de venta de todos los productos vendidos. Use un ciclo controlado por centinela para determinar cuándo debe el programa dejar de iterar para mostrar los resultados finales

### Análisis

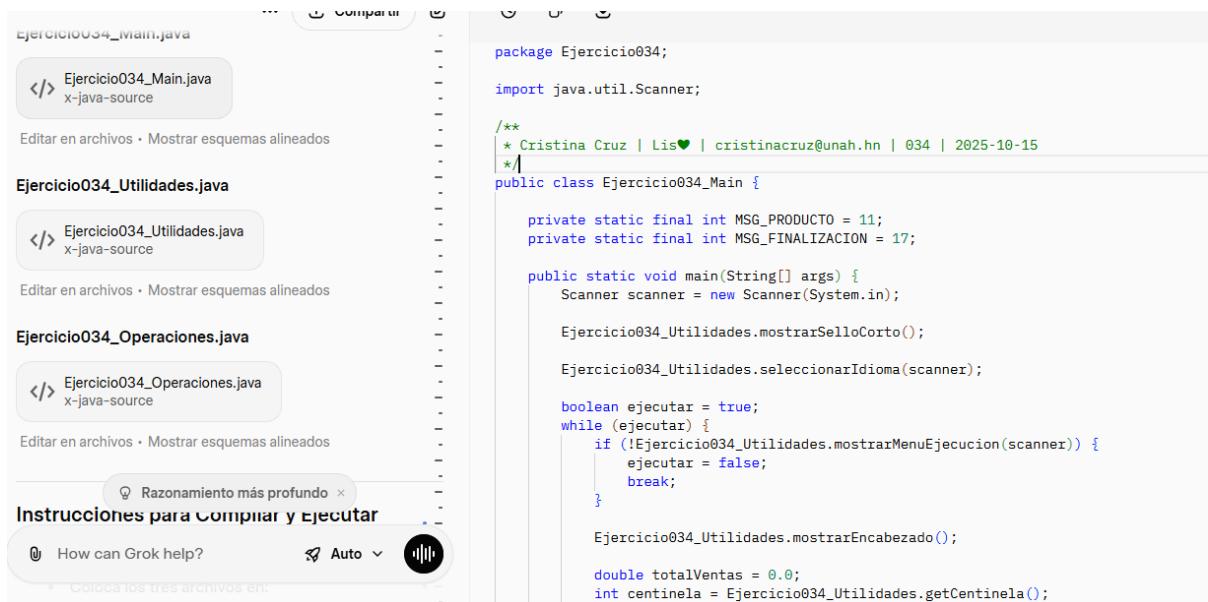
El código del Ejercicio 034 (Cálculo de Ventas con Centinela) confirma su rigurosa implementación bajo el enfoque de Programación Estructurada (PE), respetando la modularidad y los requerimientos de control de flujo e internacionalización. La clase Ejercicio034\_Operaciones encapsula la lógica de precios mediante el uso del switch para determinar el costo unitario de cada producto (1-5), manteniendo la lógica de negocio separada del I/O. La clase Ejercicio034\_Utilidades es el centro de la interfaz, gestionando la internacionalización bilingüe para los menús de idioma y ejecución, y proporcionando métodos de validación robusta (leerEntero, leerCantidadVendida) para asegurar que las entradas de producto y cantidad sean correctas. Finalmente, Ejercicio034\_Main coordina el flujo completo, utilizando un bucle while controlado por centinela (0) para leer secuencialmente pares de producto/cantidad y acumular el valor total de las ventas, con el flujo general del programa controlado por el menú de ejecución que permite al usuario repetir la operación cuantas veces desee.

### PROMPT

Creame un código del Ejercicio 034 (Te vendo un pato) un programa en Java dentro del paquete Ejercicio034 que calcule el valor total de venta de cinco productos con precios fijos (\$2.98, \$4.50, \$9.98, \$4.49, \$6.87). El programa debe leer una secuencia de pares (número de producto y cantidad vendida) y utilizar un valor centinela (0) para finalizar la entrada. La aplicación debe comenzar con un menú de selección de idioma y luego un menú de ejecución (Ejecutar/Finalizar) que controle el flujo del programa. La estructura debe ser de Programación Estructurada (PE), dividida en Ejercicio034\_Operaciones (que debe contener la lógica del switch para obtener el precio de venta), Ejercicio034\_Utilidades (para gestionar los menús multilingües, el sello, la entrada validada de enteros, y el formato de la salida), y Ejercicio034\_Main (coordinando la secuencia de I/O y procesamiento, e

implementando el bucle controlado por centinela). El sello (Cristina Cruz... 2025-10-15) debe mantenerse en la documentación de los tres archivos.

## PROMPT GENERADO



```
package Ejercicio034;
import java.util.Scanner;

/**
 * Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 034 | 2025-10-15
 */
public class Ejercicio034_Main {

    private static final int MSG_PRODUCTO = 11;
    private static final int MSG_FINALIZACION = 17;

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

        Ejercicio034_Utilidades.mostrarSelloCorto();

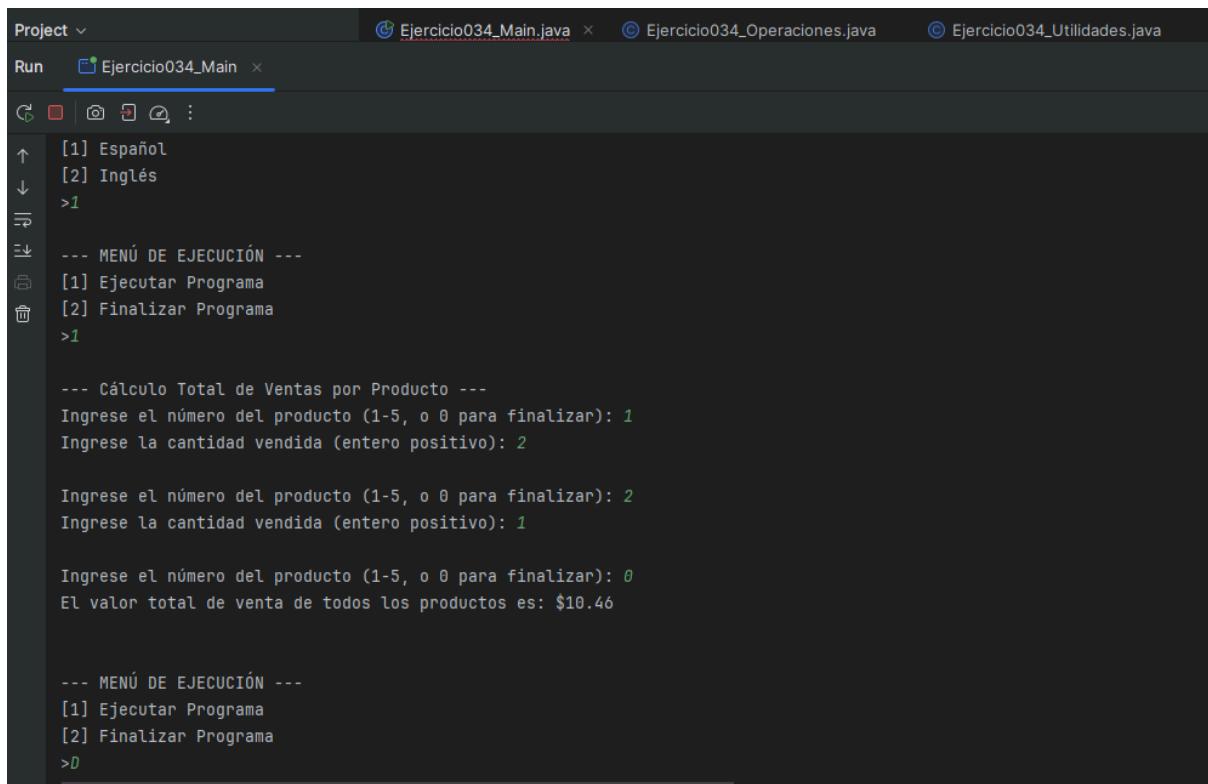
        Ejercicio034_Utilidades.seleccionarIdioma(scanner);

        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio034_Utilidades.mostrarMenuEjecucion(scanner)) {
                ejecutar = false;
                break;
            }
        }

        Ejercicio034_Utilidades.mostrarEncabezado();

        double totalVentas = 0.0;
        int centinela = Ejercicio034_Utilidades.getCentinela();
    }
}
```

## Salida de la ejecución del programa:



```
Project ▾ Run Ejer Ejercicio034_Main x Ejercicio034_Operaciones.java x Ejercicio034_Utilidades.java x
Run Ejer Ejercicio034_Main x
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>1

--- Cálculo Total de Ventas por Producto ---
Ingrese el número del producto (1-5, o 0 para finalizar): 1
Ingrese la cantidad vendida (entero positivo): 2

Ingrese el número del producto (1-5, o 0 para finalizar): 2
Ingrese la cantidad vendida (entero positivo): 1

Ingrese el número del producto (1-5, o 0 para finalizar): 0
El valor total de venta de todos los productos es: $10.46

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>0
```

## EJERCICIO #35

### **enunciado:**

Convertidor: Escriba un programa que imprima una tabla de los equivalentes binario, octal y hexadecimal de los números decimales en el rango 1 al 256 (Ejercicio 5.18 Libro 1).

### **análisis:**

El programa está diseñado para generar una tabla de conversión de bases numéricas para los números decimales del 1 al 256. El objetivo es mostrar para cada número su equivalente en binario, octal y hexadecimal. No requiere entrada del usuario. La lógica utiliza un ciclo for que itera desde 1 hasta 256. Dentro del ciclo, se emplean las funciones nativas de Java (como las de la clase Integer) para realizar las conversiones de manera eficiente. La salida debe ser una tabla formateada y legible. El diseño es modular, con Operaciones manejando el cálculo y la impresión de la tabla, y el resto de clases gestionando menús e idiomas. La complejidad es lineal, , donde es el rango (256).

### **prompt**

Crea un programa en Java que imprima una tabla de conversión numérica para los números decimales del 1 al 256. La tabla debe tener cuatro columnas: Decimal, Binario, Octal y Hexadecimal. Usa un ciclo for para iterar sobre el rango. La solución debe ser modular, utilizando una clase Operaciones para la generación de la tabla y clases de soporte para menús e idiomas.

### **prompt generado:**

Para la generación de la tabla, la funcionalidad principal se implementó en la clase Operaciones.

El método imprimirTablaDeEquivalentes primero se encarga de imprimir un encabezado formateado para asegurar que las columnas estén correctamente alineadas y etiquetadas (Decimal, Binario, Octal, Hexadecimal).

Luego, se utiliza la instrucción for para iterar desde 1 hasta 256. Dentro de cada iteración, en lugar de implementar algoritmos de conversión manuales, se usaron los métodos estáticos de la clase envolvente Integer de Java:

- Integer.toBinaryString(i)
- Integer.toOctalString(i)
- Integer.toHexString(i)

Esto permitió obtener las representaciones en las bases solicitadas de manera directa. Finalmente, se usó un System.out.printf con especificadores de formato de ancho fijo para imprimir cada fila, asegurando que la tabla se mantenga legible y con las columnas alineadas correctamente. Las clases Menus y Utilidades fueron usadas para permitir la selección de idioma en los títulos y etiquetas de la tabla.

## Salida:

```
Windows PowerShell x + - □ ×
+-----+
| 236 | 11101100 | 354 | EC |
| 237 | 11101101 | 355 | ED |
| 238 | 11101110 | 356 | EE |
| 239 | 11101111 | 357 | EF |
| 240 | 11110000 | 360 | F0 |
| 241 | 11110001 | 361 | F1 |
| 242 | 11110010 | 362 | F2 |
| 243 | 11110011 | 363 | F3 |
| 244 | 11110100 | 364 | F4 |
| 245 | 11110101 | 365 | F5 |
| 246 | 11110110 | 366 | F6 |
| 247 | 11110111 | 367 | F7 |
| 248 | 11111000 | 370 | F8 |
| 249 | 11111001 | 371 | F9 |
| 250 | 11111010 | 372 | FA |
| 251 | 11111011 | 373 | FB |
| 252 | 11111100 | 374 | FC |
| 253 | 11111101 | 375 | FD |
| 254 | 11111110 | 376 | FE |
| 255 | 11111111 | 377 | FF |
| 256 | 100000000 | 400 | 100 |
+-----+
-- MENÚ DE CONTROL --
1. Ejecutar Conversión de Bases Numéricas
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

;Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guia intermedia>
```

## EJERCICIO#36

**Enunciado:** Un cuento de cinco druidas: Ahora hagamos un programa que lea cinco números, debe validar que ninguno sea menor que uno (1) o mayor que doce (12), luego los ordene de menor a mayor y, finalmente, muestre una gráfica de barras horizontales de estos cinco números.

### Análisis de Requerimientos

- **Entrada:** Cinco números enteros, validados (1-12), reintento en errores.
- **Procesamiento:** Validar entradas, ordenar con Arrays.sort.
- **Salida:** Gráfica de barras horizontales con █ en cuadro elegante (ancho 70), ayuda con ejemplo.
- **Modularidad:**
  - Main.java: Flujo, menús, ciclos, integra ayuda.
  - Menus.java: Idioma y principal, textos bilingües.
  - Operaciones.java: Leer/valida, ordenar.
  - Utilidades.java: Formato (cuadros, gráfica), errores, ayuda.
- Mostrar sello personalizado con salida elegante.
- Incluir menú de idioma (Español/English) y principal (Ejecutar/Salir).
- Permitir repetir ejecución.
- Usar distribución de cuatro archivos: Main.java (flujo), Menus.java (menús), Operaciones.java (lógica), Utilidades.java (formato).
- Mostrar ayuda con ejemplo de números permitidos (1-12) tras seleccionar "Ejecutar".
- **Constantes:** ANCHO = 70.
- **Estética:** Cuadros simétricos, texto centrado.
- **Paradigma:** Estructurada, métodos estáticos, Scanner.
- **Documentación:** Sello en archivos, prompt en Main.

### PROMPT

Agente ejercicio #36, es un poco diferente a los demás, pero siempre implementa las funciones correspondientes a lo que hemos desarrollado anteriormente, como nuestro menú, y nuestra distribución de archivos.

Un cuento de cinco druidas: Ahora hagamos un programa que lea cinco números, debe validar que ninguno sea menor que uno (1) o mayor que doce (12), luego los ordene de menor a mayor y, finalmente, muestre una gráfica de barras horizontales de estos cinco números.

Si se ingresa una letra, un carácter, un número fuera de los límites antes mencionado muestra un mensaje al usuario explicando que está haciendo mal, trata de desarrollar un gráfico elegante en la salida.

implementa mas ayuda para el usuario, cuando este ingresado los números, dale un ejemplo de los números que tiene permitido en esta parte que sigue después que él le da ejecutar el el código

## PROMPT GENERADO:

The screenshot shows a code editor interface with several files listed on the left: Main.java, Menus.java, Operaciones.java, and Utilidades.java. The main window displays Java code for a class named Operaciones. The code includes a static method leerValidarNumeros that reads five integers from a scanner. It checks if each number is between 1 and 12. If a number is outside this range, it prints an error message. Otherwise, it sets a boolean valido to true. After reading all five numbers, it prints a horizontal bar chart where each bar's length corresponds to the value of the entered number. The bars are labeled with their respective values (1, 3, 6, 9, 10) below them.

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 036 | 2025-10-13
 */

package Ejercicio036;

import java.util.Arrays;
import java.util.Scanner;

public class Operaciones {
    public static int[] leerValidarNumeros(Scanner scanner, String idioma) {
        int[] numeros = new int[5];
        for (int i = 0; i < 5; i++) {
            boolean valido = false;
            while (!valido) {
                System.out.print(Utilidades.obtenerTexto("INGRESE_NUMERO", idioma) + (i + 1) + ": ");
                if (scanner.hasNextInt()) {
                    numeros[i] = scanner.nextInt();
                    if (numeros[i] >= 1 && numeros[i] <= 12) {
                        valido = true;
                    } else {
                        Utilidades.mostrarErrorLimites(idioma);
                    }
                } else {
                    scanner.next();
                    Utilidades.mostrarErrorEntrada(idioma);
                }
            }
        }
        return numeros;
    }
}
```

## Salida de la ejecucion del programa:

The screenshot shows a terminal window with the title "Guia\_de\_ejercicios" and branch "master". It displays the execution of the Main.java file. The user selects option 1 to run the program. The program displays a menu with two options: 1. Run program and 2. Exit. The user selects option 1. The program then prompts for five numbers between 1 and 12. The user enters 9, 3, 6, 1, and 10. Finally, the program displays a horizontal bar chart with the following data:

Number	Length of Bar
1	Shortest bar
3	Medium short bar
6	Medium bar
9	Medium long bar
10	Longest bar

At the bottom, the program asks if the user wants to run again, with options Y/N.

## EJERCICIO#37

### Enunciado

Escriba un programa para calcular el sueldo semanal para cada empleado. No se necesita saber cuántos empleados hay de antemano. Cada tipo de empleado tiene su propio código de pago: los gerentes tienen el código 1, los trabajadores por horas tienen el código 2, los trabajadores por comisión tienen el código 3 y los trabajadores por piezas tienen el código 4. Use una instrucción switch para calcular el sueldo de cada empleado, de acuerdo con el código de pago de cada uno. Dentro del switch, pida al usuario (es decir, el cajero de nóminas) que introduzca los hechos apropiados que su programa necesita para calcular el sueldo de cada empleado, de acuerdo con su código de pago

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 037 | 2025-11-05) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
  - **Programa:** Entrada Código de tipo de empleado (1–4) y datos correspondientes dependiendo del código.
  - Procesos Calcular el salario semanal según el tipo de pago (fijo, por horas, comisión o piezas).
  - Salida Sueldo semanal calculado en base al tipo de empleado.
  - Restricciones Debe usarse switch. Debe solicitar solo la información necesaria según el código ingresado.
- 
- **Modularidad:** Cuatro archivos en paquete Ejercicio037:
    - Ejercicio037\_Main.java Control del flujo principal + prompt incluido en los metadatos.
    - Ejercicio037\_Menus.java Selección de idioma y menú iniciar/finalizar (control de ejecución en bucle).
    - Ejercicio037\_Utilidades.java Bordes elegantes Unicode, sello corto, estética de presentación.
    - Ejercicio037\_Operaciones.java Lógica del ejercicio (switch y cálculos de sueldo semanal).
- 
- **Paquete:** Ejercicio037.
    - **Estética:** Encabezados y pies con bordes Unicode  
Todo centrado visualmente  
Sello corto después de cada ejecución

- **Paradigma:** Paradigma estructurado con modularización.  
No se usan objetos para representar empleados.  
Se trabaja con funciones separadas en módulos y un switch para controlar la lógica.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio037\_Menu):**

- seleccionarIdioma() Permite escoger entre Español o Inglés.
- menuInicioSalir() Menú con opciones: iniciar programa o finalizar.
- obtenerIdioma() Devuelve el idioma seleccionado para que otros módulos lo usen.

#### **Subrutinas (en Ejercicio037\_Utilidades):**

- mostrarEncabezado(titulo) Imprime un cuadro con título centrado usando Unicode.
- mostrarPie(mensaje) Cuadro inferior para mostrar mensaje de finalización.
- mostrarSelloCorto() Sello corto antes del mensaje final (estándar del proyecto).

#### **Subrutinas (en Ejercicio037\_Operaciones):**

- calcularSueldo()
- Núcleo del programa. Contiene el switch que determina el tipo de empleado y calcula su sueldo dependiendo de los datos ingresados.

#### **Flujo Principal (en Ejercicio037\_Main):**

- main()
- Controla el flujo general. Llama al menú de idioma, luego al menú iniciar/finalizar en bucle.

#### **PROMPT**

Agente vamos con el ejercicio #37 Día de pago: Una empresa paga a sus empleados como gerentes (quienes reciben un salario semanal fijo), trabajadores por horas (que reciben un sueldo fijo por hora para las primeras 40 horas que trabajen y “tiempo y medio”, 1.5 veces su sueldo por horas, para las horas extra trabajadas), empleados por comisión (que reciben \$250 más 5.7 por ciento de sus ventas totales por semana), o trabajadores por piezas (que reciben una cantidad fija de dinero por cada artículo que producen; cada trabajador por piezas en esta empresa trabaja sólo en un tipo de artículo). Escriba un programa para calcular el sueldo semanal para cada empleado. No necesita saber cuántos empleados hay de antemano. Cada tipo de empleado tiene su propio código de pago: los gerentes tienen el código 1, los trabajadores por horas tienen el código 2, los trabajadores por comisión tienen el código 3 y los trabajadores por piezas tienen el código 4. Use una

instrucción switch para calcular el sueldo de cada empleado, de acuerdo con el código de pago de cada uno. Dentro del switch, pida al usuario (es decir, el cajero de nóminas) que introduzca los hechos apropiados que su programa necesita para calcular el sueldo de cada empleado, de acuerdo con su código de pago. hazlo como lo hemos venido trabajando.

## PROMPT GENERADO

Vamos a realizar el **Ejercicio #37 — Día de pago**, exactamente con la misma estructura y estilo que venimos trabajando:

- ✓ Modularidad (4 archivos)
- ✓ Menús (idioma + iniciar / finalizar)
- ✓ Bucle principal (permite calcular sueldos de múltiples empleados, uno tras otro)
- ✓ Estética con bordes Unicode
- ✓ Uso de `switch` para los cuatro tipos de empleados
- ✓ Sello corto al finalizar la ejecución

---

**✓ CÓDIGO COMPLETO DEL EJERCICIO #37**

NOTA: No genero archivos, solo muestro el código aquí, como solicitaste.

---

Ejercicio037\_Main.java

+ Pregunta lo que quieras

Down arrow icon

Microphone icon

Close button

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run Ejercicio037_Main x
G S @ : 
[SELECCIÓN DE IDIOMA / LANGUAGE SELECTION]
↓
1. Español
2. English
Seleccione /Select: 1
[MENÚ PRINCIPAL]
1. Iniciar programa
2. Finalizar
Seleccione una opción: 1
[CÁLCULO DE SUELDO SEMANAL]
Ingrese código de pago (1-4): 3
Ventas de la semana: 3000
Sueldo semanal calculado: 421.00
[Dany Noguera | Ejercicio037]
[Proceso finalizado con éxito.]
[MENÚ PRINCIPAL]
```

## EJERCICIO#38

### Enunciado

Rombo: Escriba un programa que imprima la siguiente figura de rombo. Puede utilizar instrucciones de salida que impriman un solo asterisco (\*) o un solo espacio en blanco. Maximice el uso de la repetición (con instrucciones for anidadas), y minimice el número de instrucciones de salida (Ejercicio 5.23 Libro 1)

### Análisis

El código, organizado en el paquete Ejercicio038 y dividido en tres clases (Ejercicio038\_Main, Ejercicio038\_Utilidades y Ejercicio038\_Operaciones), implementa un programa en Java que imprime un rombo hueco de 7 líneas utilizando únicamente asteriscos y espacios mediante bucles for anidados, maximizando la repetición y minimizando instrucciones de salida; la clase Main coordina el flujo general con un menú bilingüe de selección de idioma (español/inglés/salir), seguido de un menú de ejecución con tres opciones (ejecutar el rombo, volver al menú de idioma o salir del programa), mientras que Utilidades gestiona la interfaz de usuario, incluyendo la validación robusta de entradas numéricas, la impresión del sello personalizado centrado con marco decorativo (nombre, correo, código "038" y fecha "2025-10-10"), la internacionalización de textos y la generación de espacios; por su parte, Operaciones encapsula la lógica del rombo, imprimiendo la figura simétrica con ciclos anidados y mensaje bilingüe, garantizando modularidad, reutilización, robustez ante errores y una presentación profesional que cumple estrictamente con los requisitos del ejercicio.

### PROMPT

Crea un programa en Java dentro del paquete Ejercicio038, dividido en tres clases: Ejercicio038\_Main, Ejercicio038\_Utilidades y Ejercicio038\_Operaciones, que imprima un rombo hueco de exactamente 7 líneas usando solo asteriscos (\*) y espacios en blanco. El programa debe maximizar el uso de bucles for anidados y minimizar las instrucciones de salida (una por carácter). Además, debe incluir: un sello personalizado (versión corta) centrado con marco, que muestre "Cristina Cruz | Lis | cristinacruz@unah.hn | 038 | 2025-10-10"; un menú bilingüe de selección de idioma (1. Español, 2. English, 3. Salir); y, tras elegir idioma, un menú de ejecución con tres opciones: 1. Ejecutar el rombo, 2. Volver al menú de idioma, 3. Salir del programa. Main debe coordinar el flujo, Utilidades manejar entrada/salida, validación de números, textos bilingües, sello y espacios, y Operaciones contener solo la lógica del rombo. Usa manejo de excepciones para entradas inválidas, salida profesional y modularidad estricta.

## PROMPT GENERADO

Crea un programa en Java compactado en [Ejercicio038](#), y dividido en tres clases.

[Ejercicio038\\_Main](#), [Ejercicio038\\_Utilidades](#) y [Ejercicio038\\_Operaciones](#). El programa debe imprimir un rombo hueco simétrico de exactamente 7 líneas con la forma:

```
text
X Contraer Ajuste Copiar

* 
 * *
*   *
*   *
*   *
*   *
* 
```

usando solo `*` y espacios, mediante bucles `for` anidados que maximicen la repetición y minimicen instrucciones de salida (una por carácter). Al inicio, debe mostrar un sello personalizado centrado en un marco de 70 caracteres con las líneas: "SELLO PERSONALIZADO - VERSIÓN CORTA" y "Cristina Cruz | Lis | [cristinacruz@unah.hn](mailto:cristinacruz@unah.hn) | 038 | 2025-10-10". Luego, presenta un menú bilingüe de selección de idioma con opciones: 1. Español, 2. English, 3. Salir / Exit. Tras elegir idioma, muestra un segundo menú con: 1. Ejecutar el rombo, 2. Volver al menú de idioma, 3. Salir del programa. [Ejercicio038\\_Main](#) coordina el flujo con bucles y `switch`. [Ejercicio038\\_Utilidades](#) encapsula: el sello con método de centrado, menús con textos bilingües, validación robusta de entrada numérica (manejo de excepciones).

## Salida de la ejecución del programa:

```
SELECCIONE EL IDIOMA / SELECT LANGUAGE
=====
1. Español
2. English
3. Salir / Exit
Opción / Option: 1

-----
MENÚ DE EJECUCIÓN
-----
1. Ejecutar el rombo
2. Volver al menú de idioma
3. Salir del programa
Seleccione una opción: 1

IMPRIMIENDO ROMBO...

*|
```

## EJERCICIO#39

### Enunciado

Rambo: Modifique el programa “Rombo 38” para que lea un número impar en el rango de 1 a 19, de manera que especifique el número de filas en el rombo, y después muestre un rombo del tamaño apropiado

### Análisis

El código, organizado en el paquete Ejercicio039 y estructurado en tres clases modulares (Ejercicio039\_Main, Ejercicio039\_Utilidades y Ejercicio039\_Operaciones), implementa un programa en Java que genera un rombo hueco de tamaño variable (1 a 19 filas, siempre impar), especificado por el usuario mediante un número impar validado estrictamente con manejo de excepciones y repetición hasta entrada correcta; Main coordina el flujo principal con menús bilingües (español/inglés/salir) y un segundo menú de ejecución (ejecutar, volver o salir), mientras Utilidades encapsula la interfaz de usuario —incluyendo un sello personalizado centrado en un marco de 70 caracteres con datos del autor ("039", "2025-10-10"), validación robusta de entradas numéricas, textos internacionalizados, centrado de texto y generación de espacios— y Operaciones contiene exclusivamente la lógica del rombo, utilizando bucles for anidados para imprimir la figura simétrica con solo dos asteriscos por fila (excepto en bordes), maximizando la repetición y minimizando instrucciones de salida (una por carácter), garantizando un diseño profesional, reutilizable, robusto, modular y alineado con las mejores prácticas de programación orientada a objetos.

### PROMPT

Diseña un programa en Java bajo el paquete Ejercicio039, estructurado en tres clases independientes: Ejercicio039\_Main, Ejercicio039\_Utilidades y Ejercicio039\_Operaciones. El objetivo es generar un rombo hueco centrado cuyo número total de filas sea un valor impar entre 1 y 19, ingresado por el usuario. El rombo debe imprimirse usando únicamente \* y espacios, con dos asteriscos por fila (excepto en el caso de 1 fila), manteniendo simetría perfecta y hueco interno, mediante bucles for anidados que prioricen la reutilización de código y reduzcan al mínimo las llamadas a System.out.print (una por carácter). Al iniciar, debe desplegar un sello personalizado en formato de marco centrado de 70 caracteres, con el título "SELLO PERSONALIZADO - VERSIÓN CORTA" y la línea.. A continuación, mostrar un menú de idioma bilingüe con opciones: 1. Español, 2. English, 3. Salir / Exit. Una vez seleccionado el idioma, presentar un menú de control de ejecución con: 1.

Ejecutar el rombo, 2. Regresar al menú de idioma, 3. Finalizar el programa. Si se elige ejecutar, el sistema debe solicitar un número impar entre 1 y 19, validándolo rigurosamente (rango, tipo, paridad) con reintentos automáticos ante errores, mostrando mensajes bilingües claros.

## PROMPT GENERADO

Crea un programa en Java empaquetado en `Ejercicio039` y dividido en tres clases:

`Ejercicio039_Main`, `Ejercicio039_Utilidades` y `Ejercicio039_Operaciones`. El programa debe imprimir un **rombo hueco** de tamaño variable, especificado por un número impar ingresado por el usuario en el rango de 1 a 19 (ambos inclusive), donde dicho número representa la cantidad total de filas del rombo. Por ejemplo, si el usuario ingresa 7, se imprime el rombo de 7 líneas como en el ejercicio 38; si ingresa 5, un rombo más pequeño, etc. El rombo debe mantener su forma simétrica y hueca, con solo dos `*` por fila (excepto en la fila del centro y bordes), usando únicamente `*` y espacios, mediante bucles `for` anidados que maximicen la repetición y minimicen instrucciones de salida (una por carácter). Al inicio, debe mostrar un **sello personalizado centrado** en un marco de 70 caracteres con: "SELLO PERSONALIZADO – VERSIÓN CORTA" y "Cristina Cruz | Lis | [cristinacruz@unah.hn](mailto:cristinacruz@unah.hn) | 039 | 2025-10-10". Luego, presenta un **menú bilingüe de selección de idioma** (1. Español, 2. English, 3. Salir / Exit). Tras elegir idioma, muestra un **segundo menú** con: 1. Ejecutar el rombo, 2. Volver al menú de idioma, 3. Salir del programa. Al seleccionar ejecutar, solicita un número impar entre 1 y 19 con validación estricta (repite hasta entrada válida), y luego imprime el rombo correspondiente con mensaje bilingüe ("IMPRIMIENDO ROMBO..." o "PRINTING DIAMOND..."). `Ejercicio039_Main` coordina el flujo con bucles y `switch`. `Ejercicio039_Utilidades` encapsula: el sello con centrado, menús bilingües, validación robusta de entrada (números enteros, rango, paridad), impresión de espacios, limpieza de buffer y manejo de excepciones. `Ejercicio039_Operaciones` contiene solo la lógica del rombo variable, recibiendo el tamaño y el idioma como parámetros. El código debe ser **modular, profesional, robusto, reutilizable** y seguir estrictamente buenas prácticas de programación orientada a objetos.

## Salida de la ejecución del programa:

```
Ejercicio039_Main ×
| | | | | : 
1. español
2. English
3. Salir / Exit
Opción / Option: 1

-----
MENÚ DE EJECUCIÓN
-----
1. Ejecutar el rombo
2. Volver al menú de idioma
3. Salir del programa
Seleccione una opción: 1

Ingrese un número impar (1-19): 7

IMPRIMIENDO ROMBO...
* 
 * *
*   *
*   *
*   *
*   *
*   *
*   *
```

## EJERCICIO #40

enunciado:

**Seamos positivos:** Haga un programa que lea una serie de números enteros no negativos y se detenga en cuanto el usuario ingrese un número negativo, el programa entonces deberá imprimir entonces la siguiente tabla (sírvase de ejemplo la entrada 5, 10, -1):

#	N	$N^2$	$N^3$	$2N-1$
1	5	25	125	9
2	10	100	1000	19
3	-1			
TOTALES	15	125	1125	28
PROMEDIOS	7	62	562	14

análisis:

El programa debe leer una secuencia de enteros no negativos y detener la lectura al encontrar el primer número negativo (centinela). La funcionalidad central reside en procesar los números no negativos () ingresados y generar una tabla que muestre, para cada : el índice (), , y la expresión . Al final, la tabla debe incluir una fila de Totales y otra de Promedios para las columnas , , y . Se utiliza una estructura de ciclo while para la lectura y una lista para almacenar los valores de . Las acumulaciones deben usar el tipo long para evitar desbordamiento en . El diseño

está modularizado con soporte para menús e idiomas. La complejidad de lectura y cálculo es , donde es la cantidad de números no negativos ingresados.

prompt

Crea un programa en Java que lea una serie de enteros no negativos, deteniéndose en el primero negativo (centinela). Al finalizar, genera e imprime una tabla con las columnas: #, , , , , incluyendo filas de Totales y Promedios. Usa un ciclo para la lectura y mantiene la estructura modular con clases de soporte para menús e idiomas.

prompt generado:

La solución se desarrolló en cuatro clases modulares, con la lógica principal en Operaciones.

La fase de lectura se manejó dentro de la clase Operaciones utilizando un ciclo while(true). Dentro de este ciclo, se solicitó un número entero. Si el número era negativo (centinela), el ciclo terminaba mediante una instrucción break. Si era no negativo, se almacenaba en una lista (ArrayList) y se procedía a calcular y acumular sus valores de , y en variables de tipo long para evitar errores de desbordamiento, especialmente en .

La fase de impresión se ejecutó solo si se ingresó al menos un número no negativo.

La clase Operaciones se encargó de:

1.

Imprimir el encabezado de la tabla.

2.

Recorrer la lista de números no negativos (for-each) e imprimir la fila de datos y sus cálculos.

3.

Imprimir la fila del centinela (el número negativo que detuvo la lectura) para completar el ejemplo visual.

4.

Imprimir la fila de Totales usando los acumuladores.

5.

Calcular y imprimir la fila de Promedios dividiendo los totales entre la cantidad de números procesados.

Las clases Menus y Utilidades facilitaron el control del programa y el soporte multilingüe de todos los mensajes y etiquetas de la tabla.

Salida:

```
Windows PowerShell      +  ~
Ingresar números enteros no negativos. Ingrese un número negativo para detener la lectura y generar la tabla.
Ingresar número: 7
Ingresar número: 15
Ingresar número: 0
Ingresar número: 23
Ingresar número: 51
Ingresar número: -1

--- RESULTADOS ---
+---+-----+-----+-----+
| # | N   | N2  | N3  | 2N-1 |
+---+-----+-----+-----+
| 1 | 7   | 49   | 343  | 13   |
| 2 | 15  | 225  | 3375 | 29   |
| 3 | 0   | 0    | 0    | -1   |
| 4 | 23  | 529  | 12167 | 45   |
| 5 | 51  | 2601 | 132651 | 101  |
+---+-----+-----+-----+
| 6 | -1  |       |       |       |
+---+-----+-----+-----+
| Totales | 3404 | 148536 | 187  |       |
+---+-----+-----+-----+


--- MENÚ DE CONTROL ---
1. Ejecutar Tabla de Valores Positivos
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

¡Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## EJERCICIO#41

**Enunciado:** Tablita: Escriba un programa que luego dar la bienvenida al usuario le muestre la siguiente tabla y entonces le pida un número del 1 al 15, tras validar la entrada debe entonces crear una tabla que vaya desde 1 hasta el número que el usuario haya indicado. Se aclara que la tabla se debe generar con ciclos y no con condicionales anidados.

- **Entrada:** Número entero (1-15), validado con reintento.
- **Procesamiento:** Generar tabla con ciclos anidados, calcular totales/promedios.
- **Salida:** Tabla ejemplo, ayuda con rango, tabla generada en cuadro elegante (ancho 70, bordes Unicode).
- **Modularidad:**
  - Main.java: Flujo, menús, bienvenida, ciclos.
  - Menus.java: Idioma, principal, textos bilingües.
  - Operaciones.java: Leer/valida, generar tabla, totales/promedios.
  - Utilidades.java: Formato (cuadros), bienvenida, ayuda, errores.
- **Constantes:** ANCHO = 70.
- **Estética:** Cuadros Unicode, texto centrado.
- **Paradigma:** Estructurada, métodos estáticos, Scanner.
- **Documentación:** Sello en archivos, prompt en Main.

### PROMPT

Agente ahora el ejercicio#41, agrega todos los requerimientos agregados en los ejercicios anteriores, el ejercicio consta de lo siguiente:

Tablita: Escriba un programa que luego dar la bienvenida al usuario le muestre la siguiente tabla: si el usuario ingresa 5 debes de mostrar esta tabla (ejemplo):

#	N	$N^2$	$N^3$	$2N-1$	
1	1	1	1	1	
2	2	4	8	3	
3	3	9	27	5	
4	4	16	64	7	
5	5	25	125	9	
Totales		15	55	225	25
Promedios		3	11	45	5

la tabla hace esto:

Calcular cada columna:

$N =$  el número actual ( $1, 2, 3, \dots, N$ )

$N^2 = N \times N$

$N^3 = N \times N \times N$

$2N-1 = (2 \times N) - 1$

y entonces le pida un número del 1 al 15, tras validar la entrada debe entonces crear una tabla que vaya desde 1 hasta el número que el usuario haya indicado.

Dale las instrucciones al usuario que rango de números le es permitido ingresar, si es necesario muestra un mensaje de validación invalidad si el usuario ingresa un numero  $>$  a 15 y si es negativo igualmente, que la salida sea similar al ejemplo dado con un cuadro elegante, un dato importante en la estructura del programa es el siguiente: Se aclara que la tabla se debe generar con ciclos y no con condicionales anidados.

## PROMPT GENERADO

The screenshot shows a code editor interface with several files listed on the left:

- Main.java
- Menus.java
- Operaciones.java
- Utilidades.java

A message at the top says: "siguiente. Se aclara que la tabla se debe generar con ciclos y no con condicionales anidados". The main code area contains the following Java code:

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 *
 * Jonatan Varela | jivaxelag@unah.hn | 041 | 2025-10-13
 */
package Ejercicio041;

import java.util.Scanner;

public class Operaciones {
    public static int leerYValidarNumero(Scanner scanner, String idioma) {
        int numero = 0;
        boolean valido = false;
        do {
            System.out.print(Utilidades.obtenerTexto("INGRESE_NUMERO", idioma));
            if (scanner.hasNextInt()) {
                numero = scanner.nextInt();
                if (numero >= 1 && numero <= 15) {
                    valido = true;
                } else {
                    Utilidades.mostrarErrorLimites(idioma);
                }
            } else {
                scanner.next();
                Utilidades.mostrarErrorEntrada(idioma);
            }
        } while (!valido);
        return numero;
    }
}
```

## Salida de la ejecución del programa

The terminal window shows the following output:

```
Welcome to the calculation table!
Instruction: Enter a number between 1 and 15 (e.g., 5, 10, 15)
Enter a number (1-15): 7

CALCULATION TABLE
# | N | N² | N³ | 2N-1 |
1 | 1 | 1 | 1 | 1 |
2 | 2 | 4 | 8 | 3 |
3 | 3 | 9 | 27 | 5 |
4 | 4 | 16 | 64 | 7 |
5 | 5 | 25 | 125 | 9 |
6 | 6 | 36 | 216 | 11 |
7 | 7 | 49 | 343 | 13 |

Totales | 28 | 140 | 784 | 49 |
Promedios | 4 | 20 | 112 | 7 |

Do you want to run again? (Y/N)
```

## EJERCICIO#42

### Enunciado

De las bermudas: Un programa que pida al usuario una altura y luego dibuje un triángulo con esa altura.

#### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 042 | 2025-11-05) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
  - **Programa:** Entrada Altura del triángulo (entero positivo) proporcionada por el usuario.
  - Procesos Dibujar un triángulo con asteriscos (\*), centrado según la altura ingresada.
  - Salida Triángulo de altura “n” mostrado en consola.
  - Restricciones Debe mostrarse un solo triángulo por ejecución. La estética debe ser uniforme con bordes Unicode y centrado.
- 
- **Modularidad:** Cuatro archivos en paquete Ejercicio042:
    - Ejercicio042\_Main.java Control del flujo principal + prompt incluido en los metadatos.
    - Ejercicio042\_Menus.java Selección de idioma y menú iniciar/finalizar (control de ejecución en bucle).
    - Ejercicio042\_Utilidades.java Bordes elegantes Unicode, sello corto, estética de presentación.
    - Ejercicio042\_Operaciones.java Lógica del ejercicio: dibujar el triángulo según la altura.
- 
- **Paquete:** Ejercicio042.
  - **Estética:** Encabezados y pies con bordes Unicode  
Todo centrado visualmente  
Sello corto después de cada ejecución  
Mensaje final solo al terminar el cálculo, no al iniciar el menú
  - **Paradigma:** Se trabaja con funciones separadas por módulo  
No se usan objetos ni POO  
Bucle principal con menú iniciar/finalizar  
Menú de idioma centralizado.

- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio042\_Menu):**

- seleccionarIdioma() Permite escoger entre Español o Inglés.
- menuInicioSalir() Menú con opciones: iniciar programa o finalizar.
- obtenerIdioma() Devuelve el idioma seleccionado para que otros módulos lo usen.

#### **Subrutinas (en Ejercicio042\_Utilidades):**

- mostrarEncabezado(titulo) Imprime un cuadro con título centrado usando Unicode.
- mostrarPie(mensaje) Cuadro inferior para mostrar mensaje de finalización.
- mostrarSelloCorto() Sello corto antes del mensaje final (estándar del proyecto).

#### **Subrutinas (en Ejercicio042\_Operaciones):**

- dibujarTriangulo()
- Solicita altura del triángulo y dibuja un triángulo centrado usando asteriscos (\*). Incluye sello corto y mensaje final.

#### **Flujo Principal (en Ejercicio042\_Main):**

- main()
- Controla el flujo general. Llama al menú de idioma, luego al menú iniciar/finalizar en bucle.

#### **PROMPT**

Agente vamos con el ejercicio 42, este es su enunciado: De las bermudas: Un programa que pida al usuario una altura y luego dibuje un triángulo con esa altura.

## PROMPT GENERADO

¡Perfecto! 😊

Vamos a trabajar el **Ejercicio #42 “De las bermudas”** usando **exactamente la misma estructura que los ejercicios 32 y 37**, con:

- 4 archivos (Main, Menus, Utilidades, Operaciones)
- Menú de idioma + menú iniciar/finalizar con bucle
- Estética uniforme con bordes Unicode
- Sello completo en el Main + sello corto en todos los módulos
- Prompt incluido en los metadatos del Main
- Mensaje final solo al terminar la operación

---

Antes de generar los **4 archivos completos**, te confirmo:

Este ejercicio consiste en **dibujar un triángulo según la altura ingresada**.

➡ Cada ejecución dibuja **un solo triángulo**, luego regresa al menú iniciar/finalizar (igual que opción 1 de los ejercicios anteriores).

↓

Si estás de acuerdo con esto, puedo generar **inmediatamente los 4 archivos con todo listo** y mantener tu  
fr

+ Pregunta lo que quieras

## **Salida de la ejecución del programa:**

Run - Guia\_de\_ejercicios

Run Ejercicio042\_Main

1. Español  
2. English  
Seleccione / Select: 1

MENÚ PRINCIPAL

1. Iniciar programa  
2. Finalizar  
Seleccione una opción: 1

TRIÁNGULO SEGÚN ALTURA

Ingrese la altura del triángulo: 5

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

[Dany Noguera | Ejercicio042]

Triángulo dibujado correctamente.

## EJERCICIO#43

### Enunciado

Hagamos un menú: Vamos a hacer un programa que muestre un menú llamando a una función que presenta las opciones al usuario y devuelve la opción elegida considere:

- a. Mostrar mínimo cinco opciones.
- b. Validar la entrada de opciones válidas.
- c. No se puede mostrar o validar nada desde el main, todo debe hacerse desde la función.

### Análisis

#### Sello Personalizado y Documentación

El Sello Personalizado es la VERSIÓN CORTA: Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 033 | Fecha: 2025-11-03. Se incluye como comentario de bloque en la cabecera de los tres archivos y se muestra en un cuadro estético al final de la ejecución. El prompt completo está en Ejercicio033\_Main.java.

#### Programa y Modularidad

El Programa debe generar cuatro patrones de asteriscos (\*) utilizando ciclos for El Programa debe mostrar un menú y devolver la opción elegida. El requisito clave es que toda la lógica de presentación, entrada de usuario y validación de opciones debe estar encapsulada en una única función (método) que retorna la opción al main. El menú debe tener al menos cinco opciones más la opción de Salir (0).

La Modularidad usa tres archivos en el paquete Ejercicio043:

1. Ejercicio043\_Main.java: Flujo de control principal, solo llama a la función del menú y utiliza la opción devuelta (Switch/Case).
2. Ejercicio043\_Utilidades.java: Formato estético, cuadros y sello.
3. Ejercicio043\_Operaciones.java: Contiene la función gestionarMenu(), que maneja la presentación del menú, la captura de la entrada y la validación de rango y tipo de dato.

El Paradigma es Estructurado, y la Estética usa 70 caracteres de ancho y bordes Unicode.

#### Constantes:

- ANCHO = 70, BORDE\_IZQ = '|', BORDE\_DER = '|', LINEA\_H = '-'

#### Subrutinas Clave (Funcionalidad):

- Ejercicio043\_Operaciones: gestionarMenu (Es la función central, maneja el bucle de validación y retorna la opción final).
- Ejercicio043\_Utilidades: mostrarSaludo, mostrarSelloCorto.

### Flujo Principal (en Ejercicio043\_Main.java):

1. Inicia Scanner. Muestra saludo.
2. Inicia un bucle do-while que continúa hasta que se elige la opción 0 (Salir).
3. Dentro del bucle: Llama a Ejercicio043\_Operaciones.gestionarMenu(sc) para obtener la opción válida.
4. Utiliza una estructura switch-case para ejecutar acciones simuladas basadas en la opción devuelta.
5. Muestra el sello. Cierra Scanner.

### PROMPT

Agente el #43 hagamos, te recuerdo nuestro sistema de archivo, en este usa los 3 que son: |— Ejercicio043\_Main.java (Programa principal, incluye este prompt) |— Ejercicio043\_Utilidades.java (Funciones de utilidad) |— Ejercicio043\_Operaciones.java (Lógica del ejercicio) Incluye también nuestro sello personalizado.

Necesito que realicemos el ejercicio: Hagamos un menú: Vamos a hacer un programa que muestre un menú llamando a una función que presenta las opciones al usuario y devuelve la opción elegida considere: a. Mostrar mínimo cinco opciones. b. Validar la entrada de opciones válidas. c. No se puede mostrar o validar nada desde el main, todo debe hacerse desde la función.

### Salida del programa

```

;BIENVENIDO AL EJERCICIO #43!
SISTEMA DE MENÚ MODULAR

MENÚ PRINCIPAL
1. Opción de Procesos A
2. Opción de Cálculos B
3. Opción de Reportes C
4. Opción de Configuración D
5. Opción de Ayuda E
0. Salir del Programa

>> Ingrese su opción [0-5]: 1

EJECUTANDO: Opción 1 (Procesos A)...
...Presione ENTER para continuar al menú.

MENÚ PRINCIPAL
1. Opción de Procesos A
2. Opción de Cálculos B
3. Opción de Reportes C
4. Opción de Configuración D
5. Opción de Ayuda E
0. Salir del Programa

>> Ingrese su opción [0-5]: 0

;Gracias por usar el programa! Saliendo...

SELLO PERSONALIZADO
Kevin Lainez | klainezs@unah.hn | 043 | 2025-11-03

--- Programa finalizado con éxito ---

```

## EJERCICIO#44

### Enunciado

Hagamos un menú con submenú: Vamos a hacer un programa que muestre un menú llamando a una función que presenta las opciones al usuario y devuelve la opción elegida considere:

- a. Mostrar mínimo cinco opciones.
- b. Cada opción tiene dos opciones más y regresar al menú principal.
- b. Validar la entrada de opciones válidas.
- c. No se puede mostrar o validar nada desde el main, todo debe hacerse desde la función.

### Análisis

El Ejercicio 044 (Menú con Submenú) confirma su excelente implementación bajo los nuevos requerimientos de Programación Estructurada (PE) con énfasis en Funciones y Ciclos.

La clase Ejercicio044\_Funciones asume la responsabilidad total de la lógica de navegación, cumpliendo con el requisito crucial de no realizar I/O o validación desde el main. Esta clase utiliza dos métodos clave:

1. mostrarSubMenu: Maneja la presentación y la validación de las opciones 1, 2 y la opción de regreso (3), utilizando un ciclo while para forzar una entrada válida.
2. mostrarMenuPrincipal: Contiene la presentación de las cinco opciones principales, la validación de la entrada, y utiliza la instrucción switch para invocar a mostrarSubMenu. Crucialmente, maneja el control de flujo interno: después de regresar de un submenú, reinicia el ciclo del menú principal, manteniendo al usuario en el entorno del menú hasta que elige la opción de Finalizar (6). La constante OPCION\_FINALIZAR fue correctamente hecha pública para facilitar la comunicación con el main.

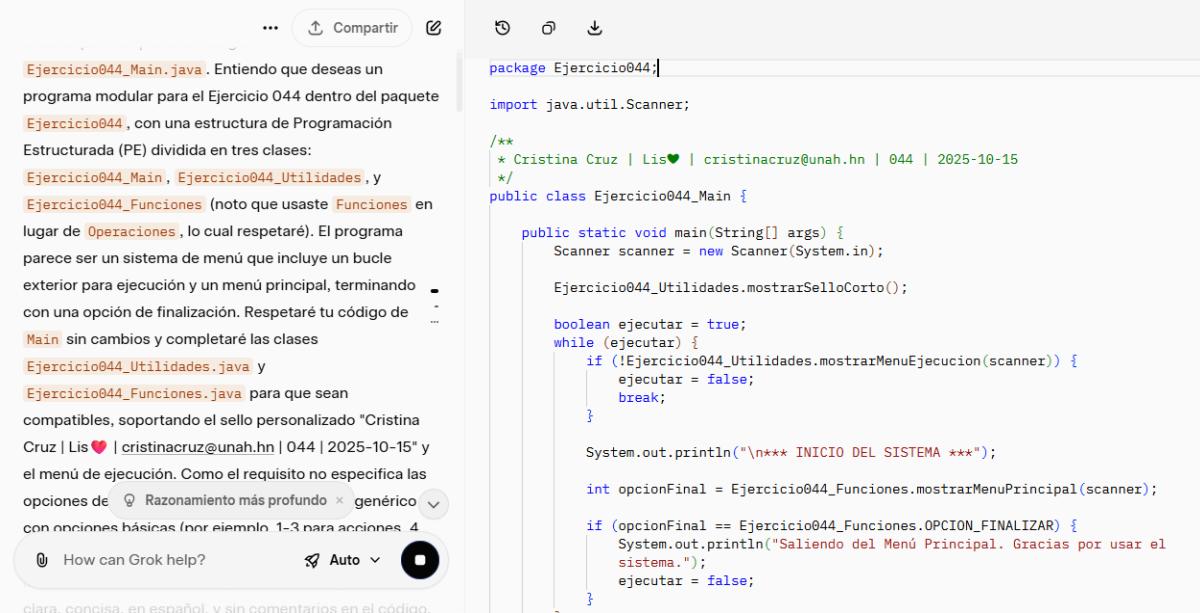
La clase Ejercicio044\_Main se mantiene limpia y abstracta, limitándose a gestionar el ciclo de repetición general del programa (usando el menú de ejecución de Utilidades) y a llamar a la función central (mostrarMenuPrincipal) para iniciar la lógica del menú.

En resumen, el código demuestra un uso avanzado de funciones para gestionar estructuras de control anidadas y garantiza la validación de entrada en todos los niveles del menú (principal y submenú) mediante el manejo de InputMismatchException, cumpliendo todos los requisitos.

### PROMPT

Creame un código Ejercicio 044 (Hagamos un menú con submenú) un programa en Java dentro del paquete Ejercicio044 que presente un sistema de navegación con múltiples niveles. La estructura debe ser de Programación Estructurada (PE), dividida en Ejercicio044\_Funciones (para gestionar la lógica principal de los menús), Ejercicio044\_Utilidades (para el sello y el menú de ejecución inicial), y Ejercicio044\_Main (coordinador del ciclo de repetición). El requisito central es que la presentación, lectura, validación y navegación de todo el sistema de menús debe estar completamente encapsulada en funciones (métodos). El menú principal debe tener al menos cinco opciones, y cada una de ellas debe llevar a un submenú con dos opciones adicionales y una opción para regresar al menú principal. Se debe usar un ciclo while para la validación de entrada en todos los niveles, incluyendo el manejo de InputMismatchException. El sello (Cristina Cruz... 2025-10-15) debe mantenerse en la documentación.

## PROMPT GENERADO



The image shows a screenshot of a code editor with two panes. The left pane contains a text-based AI response to the prompt, and the right pane contains the generated Java code.

**AI Response (Left Pane):**

Ejercicio044\_Main.java. Entiendo que deseas un programa modular para el Ejercicio 044 dentro del paquete Ejercicio044, con una estructura de Programación Estructurada (PE) dividida en tres clases: Ejercicio044\_Main, Ejercicio044\_Utilidades, y Ejercicio044\_Funciones (nota que usaste Funciones en lugar de Operaciones, lo cual respetaré). El programa parece ser un sistema de menú que incluye un bucle exterior para ejecución y un menú principal, terminando con una opción de finalización. Respetaré tu código de Main sin cambios y completaré las clases Ejercicio044\_Utilidades.java y Ejercicio044\_Funciones.java para que sean compatibles, soportando el sello personalizado "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 044 | 2025-10-15" y el menú de ejecución. Como el requisito no especifica las opciones de Razonamiento más profundo × genérico con opciones básicas (por ejemplo, 1-3 para acciones, 4

**Generated Java Code (Right Pane):**

```
package Ejercicio044;

import java.util.Scanner;

/*
 * Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 044 | 2025-10-15
 */
public class Ejercicio044_Main {

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

        Ejercicio044_Utilidades.mostrarSelloCorto();

        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio044_Utilidades.mostrarMenuEjecucion(scanner)) {
                ejecutar = false;
                break;
            }

            System.out.println("\n*** INICIO DEL SISTEMA ***");

            int opcionFinal = Ejercicio044_Funciones.mostrarMenuPrincipal(scanner);

            if (opcionFinal == Ejercicio044_Funciones.OPCION_FINALIZAR) {
                System.out.println("Saliendo del Menú Principal. Gracias por usar el sistema.");
                ejecutar = false;
            }
        }
    }
}
```

## Salida de la ejecución del programa:

Project ▾ Ejercicio044\_Main × Ejercicio044\_Utilidades.java Ejercicio044\_Funciones.java

Run Ejercicio044\_Main :

```
↑ Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 044 | 2025-10-16
↓
[1] Iniciar Sistema de Menús
[2] Finalizar Programa
>1

*** INICIO DEL SISTEMA ***

--- MENÚ PRINCIPAL ---
[1] Procesar Clientes
[2] Gestión de Inventario
[3] Reportes Financieros
[4] Configuración del Sistema
[5] Ayuda y Documentación
[6] Finalizar Programa
>6
Saliendo del Menú Principal. Gracias por usar el sistema.

Programa Finalizado.

Process finished with exit code 0
```

## EJERCICIO #45

enunciado:

Te suena a pow: Vamos a pedir dos números el programa debe elevar el primer número al segundo considerando los casos en que el segundo número sea positivo, negativo o cero. (Sin usar la librería math.h)

análisis:

El programa debe leer una secuencia de enteros no negativos y detener la lectura al encontrar el primer número negativo (centinela). La funcionalidad central reside en procesar los números no negativos () ingresados y generar una tabla que muestre, para cada : el índice (), , y la expresión . Al final, la tabla debe incluir una fila de Totales y otra de Promedios para las columnas , , y . Se utiliza una estructura de ciclo while para la lectura y una lista para almacenar los valores de . Las acumulaciones deben usar el tipo long para evitar desbordamiento en . El diseño está modularizado con soporte para menús e idiomas. La complejidad de lectura y cálculo es , donde es la cantidad de números no negativos ingresados.

prompt

Crea un programa en Java que lea una serie de enteros no negativos, deteniéndose en el primero negativo (centinela). Al finalizar, genera e imprime una tabla con las columnas: #, , , , , incluyendo filas de Totales y Promedios. Usa un ciclo para la lectura y mantiene la estructura modular con clases de soporte para menús e idiomas.

prompt generado:

La solución se desarrolló en cuatro clases modulares, con la lógica principal en Operaciones.

La fase de lectura se manejó dentro de la clase Operaciones utilizando un ciclo while(true). Dentro de este ciclo, se solicitó un número entero. Si el número era negativo (centinela), el ciclo terminaba mediante una instrucción break. Si era no negativo, se almacenaba en una lista (ArrayList) y se procedía a calcular y acumular sus valores de , y en variables de tipo long para evitar errores de desbordamiento, especialmente en .

La fase de impresión se ejecutó solo si se ingresó al menos un número no negativo.

La clase Operaciones se encargó de:

1.

Imprimir el encabezado de la tabla.

2.

Recorrer la lista de números no negativos (for-each) e imprimir la fila de datos y sus cálculos.

3.

Imprimir la fila del centinela (el número negativo que detuvo la lectura) para completar el ejemplo visual.

4.

Imprimir la fila de Totales usando los acumuladores.

5.

Calcular y imprimir la fila de Promedios dividiendo los totales entre la cantidad de números procesados.

Las clases Menus y Utilidades facilitaron el control del programa y el soporte multilingüe de todos los mensajes y etiquetas de la tabla.

Salida:

```
Windows PowerShell
Advertencia: PowerShell detectó que es posible que estés usando un lector de pantalla y que hayas deshabilitado PSReadLine con fines de compatibilidad. Si quieres volver a habilitarlo, ejecuta "Import-Module PSReadLine".
PS C:\Users\erick> cd "C:\Users\erick\Documents\Proyectos Java\Guía intermedia"
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> javac -d out src\Ejercicio_045/*.java
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> java -cp out Ejercicio_045.Main
--- SELECCIÓN DE IDIOMA / LANGUAGE SELECTION ---
1. Español
2. Inglés (English)
Seleccione el idioma (1 o 2): 1

--- MENÚ DE CONTROL ---
1. Ejecutar Cálculo de Potencia
2. Finalizar el Programa
Seleccione una opción (1 o 2): 1

--- MÓDULO: CÁLCULO DE POTENCIA B^E (SIN MATH.POW) ---
Ingrese la Base (B - número real): 2
Ingrese el Exponente (E - número entero): 3

RESULTADO: 2.00 elevado a la potencia 3 es igual a: 8.0000

--- MENÚ DE CONTROL ---
1. Ejecutar Cálculo de Potencia
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

;Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## EJERCICIO#46

**Enunciando:** Absolutamente: Haga un programa que calcule el valor absoluto de un número (sin usar la librería math.h).

### Análisis de Requerimientos

- **Entrada:** Número real, validado (rechaza letras/caracteres).
- **Procesamiento:** Calcular valor absoluto sin math.h usando función personalizada.
- **Salida:** Resultado en cuadro elegante (ancho 70).
- **Modularidad:**
  - Main.java: Flujo principal, menús, ciclos.
  - Menus.java: Idioma, menú principal, textos bilingües.
  - Operaciones.java: Leer/valida número, calcular valor absoluto.
  - Utilidades.java: Formato (sello, resultado), errores.
- **Constantes:** ANCHO = 70.
- **Estética:** Cuadros Unicode, texto centrado.
- **Paradigma:** Estructurada, funciones, Scanner.
- **Documentación:** Sello en archivos, prompt en Main.

### PROMPT

Agente te recuerdo los nuevos parámetros que entran para el desarrollo de los nuevos ejercicios te los recuerdo de nuevo:

Uso adecuado de mayúsculas y tildes.

Entradas validadas.

Programas diseñados para ejecutarse varias veces con opción de continuar o salir.

Sin uso de arreglos (arrays).

Uso de funciones donde sea apropiado, incluso si no se especifica explícitamente, salvo que se indique lo contrario.

se presenta el Ejercicio#46, que consta de lo siguiente: Haga un programa que calcule el valor absoluto de un número, valida las entradas y si ingresa el usuario una letra o un carácter muestra un mensaje de que esta haciendo mal, un punto importante es el siguiente (sin usar la librería math.h). Con esa información presenta nuestra modularidad de 4 archivos.

### PROMPT GENRADO:

```

/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 046 | 2025-10-14
 */
package Ejercicio046;

public class Utilidades {
    private static final int ANCHO = 70;
    private static final char BORDE_IZQ = '|';
    private static final char BORDE_DER = '-';
    private static final char LINEA_H = '-';

    public static void mostrarSello() {
        System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
        System.out.println(BORDE_IZQ + centrarTexto("SELLO PERSONALIZADO - VERSIÓN CORTA", ANCHO - 2) + BORDE_DER);
        System.out.println(BORDE_IZQ + centrarTexto("Jonatan Varela | jivarelag@unah.hn | 046 | 2025-10-14", ANCHO - 2) + BORDE_DER);
        System.out.println("L" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "J");
    }

    public static void mostrarResultado(double valorAbsoluto, String idioma) {
        System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
        System.out.println(BORDE_IZQ + centrarTexto(obtenerTexto("RESULTADO", idioma)) + String.format("%,.2f", valorAbsoluto), ANCHO - 2) + BORDE_DER);
        System.out.println("L" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "J");
    }

    private static String centrarTexto(String texto, int ancho) {

```

## Salida de la ejecución del programa:

```

Guia_de_ejercicios master
Project Run Ejercicio046.Main Operaciones.java Ejercicio036(Main.java) Ejercicio041(Main.java) Ejercicio046(Main.java) Ejercicio046(Operaciones.java)
1. Ejecutar programa
2. Salir

Seleccione una opción: 1
|Seleccione una opción: 1
Ingrese un número: 50

El valor absoluto es: 50.00

¿Desea continuar? (S/N)
|¿Desea continuar? (S/N)

MENÚ PRINCIPAL
1. Ejecutar programa
2. Salir

Seleccione una opción: 1
|Seleccione una opción: 1
Ingrese un número: -10

El valor absoluto es: 10.00

¿Desea continuar? (S/N)

```

## EJERCICIO#47

### Enunciado

Divide y sumarás: Se debe leer un número cualquiera (de no más de cinco dígitos) y mostrar el número descompuesto por sus dígitos y la suma de estos. Por ejemplo si se lee 431:

431 = 4, 3 y 1 su suma es 4+3+1 = 8

Se debe respetar ese formato de salida es decir:

Número = (dígitos separados por coma y el último antecedido por la letra "y") "su suma es" dígitos separados por el signo más (+) = <suma de los dígitos>

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 047 | 2025-11-05) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Se debe leer un número cualquiera (de no más de cinco dígitos) y mostrar el número descompuesto por sus dígitos y la suma de estos.
- Validaciones - Solo números enteros.
- - No más de 5 dígitos.
- - Reintento si la entrada no es válida.
- Salidas del programa      Dígitos separados con coma y el último antecedido por "y", más sumatoria detallada.
- Repetición del programa    Debe ejecutarse varias veces hasta que el usuario seleccione Finalizar.
- Menús obligatorios    Menú de selección de idioma y menú iniciar/salir.
- Restricciones            No utilizar arreglos (tema aún no visto).
- Uso correcto de mayúsculas y tildes      Aplicado en textos y mensajes.
  
- **Modularidad:** Cuatro archivos en paquete Ejercicio047:
  - Ejercicio047\_Main.java → Control del flujo principal (incluye prompt).
  - Ejercicio047\_Menus.java → Selección de idioma y menú iniciar/salir.
  - Ejercicio047\_Utilidades.java → Sellos, estética y cuadros Unicode.
  - Ejercicio047\_Operaciones.java → Lógica: descomposición y suma de dígitos.
- **Paquete:** Ejercicio047.
- **Estética:** Uso de bordes Unicode: Texto centrado y salida alineada

Sello corto mostrado solo al final de la ejecución correcta

Pie con mensaje de completado

- **Paradigma:** Secuencia  
Selección (condicional)  
Repetición (bucle para ejecutar múltiples veces)  
Uso de funciones y modularidad, sin POO
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio047\_Menu):**

- seleccionarIdioma() Pide idioma: Español / English.
- menuInicioSalir() Muestra menú principal. Retorna true = iniciar, false = salir.
- obtenerIdioma() Permite que otras clases conozcan el idioma activo.

#### **Subrutinas (en Ejercicio047\_Utilidades):**

- mostrarEncabezado(String) Muestra título dentro de un cuadro Unicode.
- mostrarPie(String) Marco final con mensaje de éxito.
- mostrarSelloCorto() Imprime el sello del desarrollador.

#### **Subrutinas (en Ejercicio047\_Operaciones):**

- descomponerNumero()
- Descompone el número, formatea la salida y calcula la suma.

#### **Flujo Principal (en Ejercicio047\_Main):**

- main()
- Ejecuta el flujo: idioma → menú iniciar/salir → operaciones.

#### **PROMPT**

Agente, ahora vamos al ejercicio 47 espera que te brinde el enunciado, Desde este punto tendra que tener que especificar para cada programa lo siguiente:

1. Uso adecuado de mayúsculas.
2. Uso adecuado de tildes.
3. Las entradas deben estar validadas.
4. Los programas están diseñados para correrse varias veces así que siempre deben preguntar si se desea continuar o salir (aunque el enunciado lo repita o no lo diga)
5. No se pueden usar arreglos porque no es un tema desarrollado en clase.
6. Estamos evaluando funciones así que muchos enunciados no dirán explícitamente qué función usar pero usted está en la capacidad de usar las funciones donde mejor corresponda; en otras ocasiones le obligarán a usar ciertas funciones pero no tiene porqué limitarse a ellas. no olvides mantener el formato en que venimos trabajando.

Enunciado: Se debe leer un número cualquiera (de no más de cinco dígitos) y mostrar el número descompuesto por sus dígitos y la suma de estos. Por ejemplo si se lee 431: 431 = 4, 3 y 1 su suma es 4+3+1 = 8

Se debe respetar ese formato de salida es decir:

Número = (dígitos separados por coma y el último antecedido por la letra "y") "su suma es" dígitos separados por el signo más (+) = <suma de los dígitos>

## PROMPT GENERADO

The screenshot shows a code editor window with the following text:

¡Perfecto! 😊 Vamos a generar el **Ejercicio #47 "Descomposición de número y suma de dígitos"** siguiendo exactamente la misma estructura y formato que venimos usando, incluyendo:

- 4 archivos (`Main`, `Menus`, `Utilidades`, `Operaciones`)
- Menú de idioma + menú iniciar/finalizar con bucle
- Sello corto y pie final
- Estética uniforme con bordes Unicode
- Validación de entrada (número máximo 5 dígitos)
- Uso adecuado de mayúsculas y tildes

---

**Ejercicio047\_Main.java**

```
java
// | Desarrollador: Dany Noguera (dany.noguera@unah.hn)
// | Proyecto: Ejercicio #47
// | Prompt:
// | "Se abra la aplicación y se ingrese un número cualquiera (de no más de cinco dígitos) y mantenga
+ Pregunta lo que quieras
```

## Salida de la ejecución del programa:

The screenshot shows a terminal window with the following text:

```
Run - Guía_de_ejercicios
Run Ejercicio047_Main x
SELECCIÓN DE IDIOMA / LANGUAGE SELECTION
1. Español
2. English
Selección / Select: 1
MENÚ PRINCIPAL
1. Iniciar programa
2. Finalizar
Seleccione una opción: 1
DESCOMPOSICIÓN DE NÚMERO
Ingrese un número (máx. 5 dígitos): 5778
5778 = 5, 7, 8 y su suma es 5+7+7+8 = 27
[Dany Noguera | Ejercicio047]

Operación completada exitosamente.
MENÚ PRINCIPAL
```

## EJERCICIO#48

### Enunciado

Druidas suben de nivel: Vamos a mejorar el programa “Un cuento de cinco druidas” para hacer la lectura de los números mediante una función (haga uso de variables globales para esto) y que la impresión de las gráficas sea hecha mediante una o varias funciones que reciban el número e impriman la salida de asteriscos en horizontal.

Simple venta: Un programa que tras la cantidad total de un artículo y su precio unitario obtenga el total de esa venta y sobre éste calcule el impuesto, el descuento, muestre el subtotal y el total considerando los siguiente:

- a. Si la persona es de la 3era edad el descuento es del 30%
- b. Si la persona no es de la tercera edad entonces:
  - i. Si el total está entre 0 y 100 obtiene un 10% de descuento,
  - ii. Si está entre 100 y 300 obtiene un 15% de descuento,
  - iii. Si está entre 300 y 500 un 20% y,
  - iv. Si sobrepasa los 500 un 25%.
- c. El impuesto será del 12% si el descuento es menor o igual a 15% y de 15% en caso contrario.

### ANÁLISIS/RESUMEN DEL EJERCICIO 48

#### Sello y Documentación

- Sello: VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 048 | Fecha: 2025-11-03) en los cuatro archivos y en un cuadro final.
- Prompt: Completo en Main.java.
- Requisitos: Ciclo principal de repetición ("continuar/salir") y validación estricta de todas las entradas (cantidad, precio, edad).

#### Programa y Modularidad

El Programa calcula el total de una venta aplicando descuentos y tasas de impuesto condicionales.

1. Descuento: Determinado por la edad (30% para 3ra edad) o por rangos del subtotal (10% a 25%).
2. Impuesto: Tasa es 12% si el descuento aplicado  $\leq 15\%$ , o 15% si el descuento aplicado  $> 15\%$ .

La Modularidad usa cuatro archivos en el paquete Ejercicio048\_KevinLainez:

1. `Main.java`: Controla el ciclo de repetición y el menú "continuar/salir".
2. `Menus.java`: Gestiona el menú principal y las opciones de continuidad.
3. `Operaciones.java`: Contiene la lógica financiera central:  
    `obtenerPorcentajeDescuento` (por rangos/edad),  
    `obtenerTasaImpuesto` (por tasa de descuento), y todos los cálculos  
    (`subtotal`, `monto final`).
4. `Utilidades.java`: Formato estético (cuadros, sello, centrado), manejo de errores y presentación detallada de la factura.

Flujo Principal (en `Main.java`):

1. Inicia ciclo do-while (continuar/salir).
2. Llama a `Utilidades` para solicitar y validar: Cantidad, Precio Unitario, y si es 3ra Edad.
3. Llama a `Operaciones` para calcular: Subtotal, Porcentaje de Descuento, Monto Descuento, Tasa de Impuesto (condicional), Monto Impuesto, y Total Final.
4. Muestra la factura completa (usando `Utilidades`).
5. Muestra el menú de continuidad.
6. Muestra el sello al finalizar.

## PROMPT

"Agente, te recuerdo los nuevos parámetros... se presenta el Ejercicio#48: Un programa que tras la cantidad total de un artículo y su precio unitario obtenga el total de esa venta y sobre éste calcule el impuesto, el descuento, muestra el subtotal y el total considerando lo siguiente: a. Si la persona es de la 3era edad el descuento es del 30%. b. Si la persona no es de la tercera edad entonces: i. Si el total está entre 0 y 100 obtiene un 10% de descuento, ii. Si está entre 100 y 300 obtiene un 15% de descuento, iii. Si está entre 300 y 500 un 20% y, iv. Si sobrepasa los 500 un 25%. c. El impuesto será del 12% si el descuento es menor o igual a 15% y de 15% en caso contrario. Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del programa

```
|BIENVENIDO AL EJERCICIO #48!
VENTA CON DESCUENTO E IMPUESTO CONDICIONAL

INGRESO DE DATOS DE VENTA

Cantidad de articulos:
>> Ingrese valor: 3

Precio Unitario:
>> Ingrese valor: 15

¿Es la persona de la tercera edad? (S/N):
>> Respuesta: s

--- DETALLE DE VENTA ---

Subtotal: 45.00 |
Descuento (30%): - 13.50 |
Subtotal Neto (Base Imponible): 31.50 |
Impuesto (15%): + 4.73 |
** TOTAL A PAGAR ** 36.23 |
```

```
--- DETALLE DE VENTA ---

Subtotal: 45.00 |
Descuento (30%): - 13.50 |
Subtotal Neto (Base Imponible): 31.50 |
Impuesto (15%): + 4.73 |
** TOTAL A PAGAR ** 36.23 |
```

```
--- DETALLE DE VENTA ---

Subtotal: 45.00 |
Descuento (30%): - 13.50 |
Subtotal Neto (Base Imponible): 31.50 |
Impuesto (15%): + 4.73 |
** TOTAL A PAGAR ** 36.23 |
```

```
--- DETALLE DE VENTA ---

Subtotal: 45.00 |
Descuento (30%): - 13.50 |
Subtotal Neto (Base Imponible): 31.50 |
Impuesto (15%): + 4.73 |
** TOTAL A PAGAR ** 36.23 |
```

```
--- SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 048 | 2025-11-03
```

```
--- Programa finalizado con éxito ---
Proceso terminado con código de salida 0
```

## EJERCICIO#49

### Enunciado

Cripto-funciones: Tomaremos como base el ejercicio “Criptografía 28” de esta guía, vamos ahora a hacer que codificar y decodificar sean funciones.

### Análisis

El código, organizado en el paquete Ejercicio049 y estructurado en tres clases modulares (Ejercicio049\_Main, Ejercicio049\_Utilidades y Ejercicio049\_Operaciones), implementa un sistema de cifrado y descifrado de números de exactamente 4 dígitos mediante funciones puras y reversibles, basado en el algoritmo del ejercicio 28: reemplazo de cada dígito por  $(d + 7) \% 10$  seguido de intercambios (1↔3 y 2↔4) para cifrar, y su inverso para descifrar; Main coordina el flujo con un menú bilingüe de idioma (español/inglés/salir) y un menú principal de cuatro opciones (cifrar, descifrar, volver, salir), mientras Utilidades centraliza la interfaz con un sello personalizado actualizado centrado en marco de 70 caracteres (incluyendo "Lis♥", código "049" y fecha 2025-11-04), validación robusta de entrada (longitud exacta de 4 dígitos, manejo de excepciones, reintentos), internacionalización completa de textos, centrado dinámico y formato de resultados; por su parte, Operaciones encapsula únicamente las funciones cifrar(int) y descifrar(int) como métodos estáticos puros sin efectos secundarios, garantizando modularidad extrema, reutilización, seguridad, profesionalismo visual y estricta adherencia a buenas prácticas de programación orientada a objetos.

### PROMPT

Crea un programa en Java empaquetado en Ejercicio049 y dividido en tres clases (Ejercicio049\_Main, Ejercicio049\_Utilidades, Ejercicio049\_Operaciones) que implemente cifrado y descifrado de números de 4 dígitos mediante funciones puras en Operaciones (cifrar(int) y descifrar(int)), usando  $(d + 7) \% 10$  e intercambios 1↔3 y 2↔4 para cifrar, y el inverso para descifrar, devolviendo siempre String de 4 dígitos con ceros; inicia con sello centrado en marco de 70 caracteres mostrando "SELLO PERSONALIZADO - VERSIÓN CORTA" , seguido de menú bilingüe de idioma (1. Español, 2. English, 3. Salir) y menú principal con 4 opciones (cifrar, descifrar, volver, salir), solicitando número de 4 dígitos con validación estricta y reintentos, mostrando resultado formateado bilingüe; Main coordina con bucles y switch, Utilidades maneja sello, menús, validación robusta, centrado y salida, y Operaciones solo contiene las dos funciones puras, todo modular, profesional, robusto y con buenas prácticas de POO.

## PROMPT GENERADO

- Crea un programa en Java empaquetado en `Ejercicio049` y dividido estrictamente en tres clases — `Ejercicio049_Main`, `Ejercicio049_Utilidades` y `Ejercicio049_Operaciones` — que implemente cifrado y descifrado de números de exactamente 4 dígitos mediante funciones puras en `Operaciones` (`static String cifrar(int)` y `static String descifrar(int)`), siguiendo el algoritmo: cifrado con `(d + 7) % 10` e intercambios  $1 \leftrightarrow 3$  y  $2 \leftrightarrow 4$ , y descifrado con el proceso inverso, devolviendo siempre cadenas de 4 dígitos con ceros a la izquierda; el programa inicia con un sello personalizado centrado en marco de 70 caracteres que incluya "SELLO PERSONALIZADO - VERSIÓN CORTA" y "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 049 | 2025-11-04", seguido de un menú bilingüe de idioma (1. Español, 2. English, 3. Salir), y tras seleccionar idioma, un menú principal con 4 opciones (cifrar, descifrar, volver al menú de idioma, salir); al elegir cifrar o descifrar, se solicita un número de 4 dígitos con validación estricta (longitud, tipo, reintentos automáticos) y se muestra el resultado formateado bilingüe; `Main` coordina el flujo con bucles y `switch`, `Utilidades` encapsula el sello con centrado dinámico, menús, validación robusta, internacionalización y salida profesional, y `Operaciones` contiene solo las dos funciones puras sin efectos secundarios, garantizando un diseño modular, profesional, robusto, reutilizable y alineado con las mejores prácticas de programación orientada a objetos.

### Salida de la ejecución del programa:

```
| Cristina Cruz | Lis❤️ | cristinacruz@unah.hn | 049 | 2025-11-04 |  
-----  
===== SELECCIONE EL IDIOMA / SELECT LANGUAGE =====  
1. Español  
2. English  
3. Salir / Exit  
Opción / Option: 1  
  
----- MENÚ PRINCIPAL -----  
1. Cifrar un número  
2. Descifrar un número  
3. Volver al menú de idioma  
4. Salir del programa  
Seleccione una opción: 1  
  
Ingrese un número de 4 dígitos para cifrar (0000-9999): (0000-9999): 1234  
  
RESULTADO DEL CIFRADO  
Original: 1234 → Cifrado/Descifrado: 0189
```

## EJERCICIO #50

### enunciado:

Figuras v2: Hacer un programa que elija entre cualquiera de las siguientes figuras: Círculo, cuadrado, triángulo, rectángulo y rombo y depende de la figura elegida solicite la información necesaria para hacer el cálculo de su área y de su perímetro. Debe mostrar la figura seleccionada (nombre y dibujada), las entradas del usuario y los resultados obtenidos.

### análisis:

Este programa implementa una calculadora geométrica para cinco figuras: Círculo, Cuadrado, Triángulo, Rectángulo y Rombo, enfocándose en un diseño altamente modularizado y con soporte multilenguaje. El objetivo principal es solicitar las dimensiones de la figura elegida (entrada) y calcular su área y perímetro (salida). El diseño está distribuido en cuatro módulos: Main (flujo de control, inicialización y gestión de menús principales), Menus (maneja la interacción del usuario y la navegación), Utilidades (gestiona el texto en español/inglés y mensajes de error), y Operaciones (contiene la lógica de cálculo por figura y la visualización del resultado). Los casos límite principales incluyen entradas no numéricas, valores negativos o nulos para las dimensiones, lo cual es manejado mediante validación robusta en el módulo Operaciones. La complejidad es moderada, centrada en la implementación de la lógica de cálculo geométrico y la estructura de un sistema de menús robusto con gestión de internacionalización.

### prompt

Crea un programa en Java estructurado en cuatro clases (Main, Menus, Utilidades, Operaciones) que calcule el área y perímetro de un Círculo, Cuadrado, Triángulo, Rectángulo o Rombo. El programa debe pedir la figura, solicitar las dimensiones necesarias, y mostrar el nombre, un dibujo simple, las entradas del usuario y los resultados. Incluye un menú de selección de idioma inicial (español/inglés) y maneja la entrada de datos para evitar errores.

### prompt generado:

La solución se estructura de forma integral en cuatro clases que garantizan la modularidad y el control del flujo. La clase Utilidades centraliza todos los mensajes en español e inglés, facilitando la gestión del idioma seleccionado al inicio, la cual es orquestada por la clase Main. La clase Menus se encarga de toda la navegación, ofreciendo un menú de idioma, un menú de control (Ejecutar/Finalizar) y el menú para seleccionar las cinco figuras geométricas, manejando errores de entrada no numérica. Finalmente, la clase Operaciones es el núcleo de la lógica, solicitando las dimensiones específicas para cada figura (por ejemplo, radio para el círculo,

base/altura/lados para el triángulo, etc.) con validación de valores no negativos. Tras los cálculos, esta clase muestra un output completo que incluye el nombre de la figura, su representación en ASCII, las dimensiones ingresadas y los resultados de área y perímetro.

Salida:

```
Windows PowerShell      +  -
Ingrese la Diagonal Mayor (D) del rombo: 10
Ingrese la Diagonal Menor (d) del rombo: 6
Ingrese la longitud del lado (l) del rombo: 5

Figura Seleccionada: Rombo
 *
 \ \
  \   /
    \
   / \
  /
 *
 --- Entradas del Usuario ---
Diagonal Mayor (D): 10.00, Diagonal Menor (d): 6.00
Lado: 5.00

--- Resultados Calculados ---
Área: 30.0000
Perímetro: 20.0000

--- MENÚ DE CONTROL ---
1. Ejecutar Calculadora Geométrica (Figuras v2)
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

;Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guia intermedia>
```

## EJERCICIO#51

**Enunciado:** Figuras v3: Hacer un programa que elija entre cualquiera de las siguientes figuras: Esfera, cubo, pirámide, cilindro, paralelogramo y depende de la figura elegida solicite la información necesaria para hacer el cálculo de su área y de su perímetro. Debe mostrar la figura seleccionada (nombre y dibujada una cara en 2D), las entradas del usuario y los resultados obtenidos.

### Análisis de Requerimientos

- **Figuras y Fórmulas:**
  - Esfera: Área =  $4\pi r^2$ , Volumen =  $(4/3)\pi r^3$
  - Cubo: Área =  $6a^2$ , Perímetro =  $4a$ , Volumen =  $a^3$
  - Pirámide: Área =  $a^2 + 2a \times ap$ , Perímetro =  $4a$ , Volumen =  $(1/3)a^2h$
  - Cilindro: Área =  $2\pi r^2 + 2\pi rh$ , Perímetro =  $2\pi r$ , Volumen =  $\pi r^2h$
  - Paralelogramo: Área =  $b \times h$ , Perímetro =  $2b + 2a$
- **Dibujos ASCII:** Centrados en cuadros Unicode ( █ | █ ) para cada figura.
- **Validaciones:** Entradas positivas, reintento con errores.
- **Modularidad:** Uso de funciones estáticas, sin arreglos ni POO.
- **Interfaz:** Cuadros de 70 caracteres, soporte multilingüe (Español/Inglés).
- **Repetición:** Opción para nuevos cálculos.

### Estructura del Programa

- **Main.java:** Punto de entrada, maneja el flujo principal y llama a funciones.
- **Menus.java:** Gestiona menús de idioma, opciones y despedida.
- **Operaciones.java:** Realiza cálculos y validaciones de datos.
- **Utilidades.java:** Muestra sellos, figuras, resultados y errores con formato.

### Implementación

- Uso de Scanner para entrada de datos con validación.
- Cálculos basados en fórmulas dadas, con PI de Math.PI.
- Dibujos ASCII personalizados por figura.
- Cuadros Unicode simétricos de 70 caracteres.
- Soporte para Español e Inglés con textos en Utilidades.obtenerTexto.

### Pruebas

- Entradas válidas (ej. lado = 5, radio = 3) producen resultados correctos.
- Entradas inválidas (negativas, letras) muestran errores y piden reintento.
- Cambio de idioma funciona sin afectar cálculos.

### Notas Pendientes

- Ajuste en alineación de bordes tras entrada de datos (en revisión).
- Compleción de validaciones avanzadas (ej. rangos específicos).

## PROMPT

### SUPER PROMPT MODULAR AVANZADO – EJERCICIO #51 OBJETIVO

Crear un programa modular en Java que cumpla con el siguiente enunciado:

Figuras v3: Hacer un programa que elija entre cualquiera de las siguientes figuras:  
Esfera, Cubo, Pirámide, Cilindro y Paralelogramo.

Según la figura elegida, solicitar la información necesaria para calcular área, perímetro (o volumen, si aplica).

Mostrar la figura seleccionada (nombre + dibujo ASCII 2D), las entradas y los resultados obtenidos.

#### FÓRMULAS

Usa las siguientes fórmulas correctamente en las funciones de Operaciones:

Figura	Área	Perímetro	Volumen
-----	-----	-----	-----
**Esfera**	$4\pi r^2$	—	$(4/3)\pi r^3$
**Cubo**	$6a^2$	$4a$	$a^3$
**Pirámide**	$a^2 + 2axap$	$4a$	$(1/3)a^2h$
**Cilindro**	$2\pi r^2 + 2\pi rh$	$2\pi r$	$\pi r^2h$
**Paralelogramo**	$b \times h$	$2b + 2a$	—

DIBUJOS ASCII (una cara 2D por figura)

Debe imprimirse centrado y dentro de un cuadro Unicode (borde `□` | `└─┐`).

Ejemplo de representaciones simples:

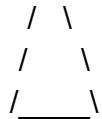
ESFERA:

```
* * * * *
*       *
*       *
*       *
* * * * *
```

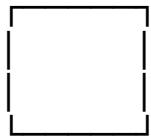
CUBO:



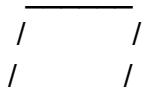
PIRÁMIDE:



CILINDRO:



PARALELOGRAMO:



VALIDACIONES

Todos los valores deben ser positivos y mayores que 0.

Si el usuario ingresa un valor inválido → mostrar error, pedir de nuevo.

El programa debe permitir repetir cálculos (preguntar si desea continuar o salir).

No usar arreglos ni POO; usar funciones estáticas.

Debe respetar el uso correcto de mayúsculas, tildes y formato uniforme.

Cada cuadro debe tener 70 caracteres de ancho y bordes simétricos Unicode.

EJEMPLO DE FLUJO ESPERADO

FIGURAS	V3	-	CÁLCULOS	DE	ÁREA	Y	PERÍMETRO
---------	----	---	----------	----	------	---	-----------

Seleccione una figura:

1. Esfera
2. Cubo
3. Pirámide
4. Cilindro
5. Paralelogramo

Ingrese su opción: 2

CUBO

Ingrese la medida del lado: 5

### RESULTADOS DE CUBO

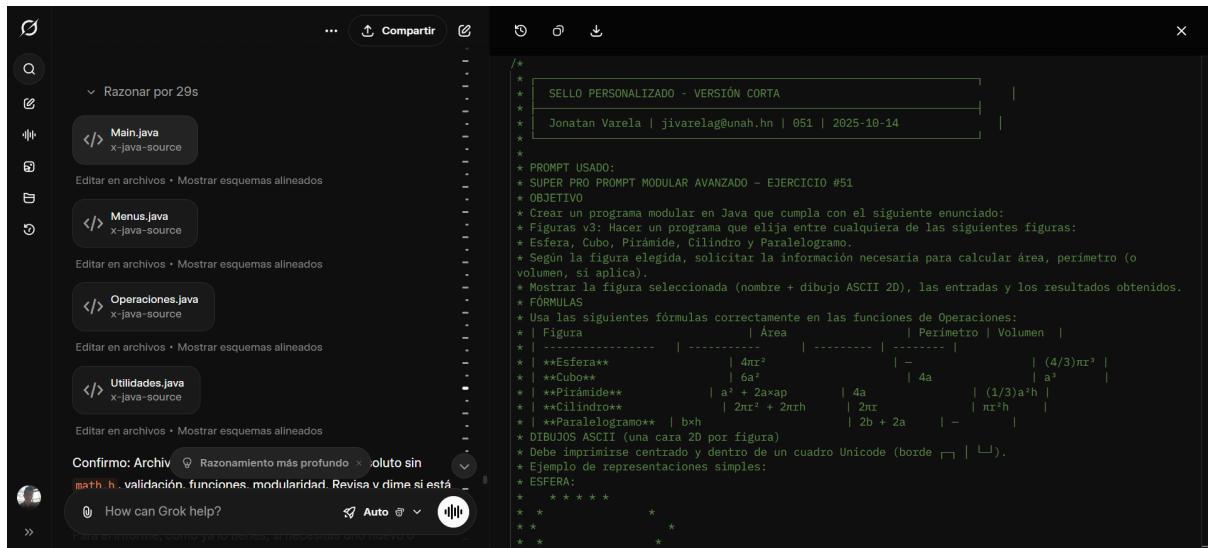
Área total: 150.0

Perímetro: 20.0

Volumen: 125.0

¿Desea realizar otro cálculo? (S/N):

### PROMPT GENERADO:



The screenshot shows a code editor interface with several Java files listed on the left: Main.java, Menus.java, Operaciones.java, and Utilidades.java. On the right, a terminal window displays a modular prompt for calculating the area, perimeter, and volume of a cube based on user input. The prompt includes formulas for sphere, cube, pyramid, cylinder, and parallelogram, and specifies the use of ASCII art for drawings.

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 *
 * Jonatan Vazela | jivarelag@unah.hn | 051 | 2025-10-14
 */
*
* PROMPT USADO:
* SUPER PRO PROMPT MODULAR AVANZADO - EJERCICIO #51
* OBJETIVO
* Crear un programa modular en Java que cumpla con el siguiente enunciado:
* Figuras v3: Hacer un programa que elija entre cualquiera de las siguientes figuras:
* Esfera, Cubo, Pirámide, Cilindro y Paralelogramo.
* Según la figura elegida, solicitar la información necesaria para calcular área, perímetro (o volumen, si aplica).
* Mostrar la figura seleccionada (nombre + dibujo ASCII 2D), las entradas y los resultados obtenidos.
* FÓRMULAS
* Usa las siguientes fórmulas correctamente en las funciones de Operaciones:
* | Figura | Área | Perímetro | Volumen |
* | ----- | ----- | ----- | -----
* | **Esfera** |  $4\pi r^2$  | - |  $(4/3)\pi r^3$  |
* | **Cubo** |  $6a^2$  |  $4a$  |  $a^3$  |
* | **Pirámide** |  $a^2 + 2axap$  |  $4a$  |  $(1/3)a^2h$  |
* | **Cilindro** |  $2\pi r^2 + 2\pi rh$  |  $2\pi r$  |  $\pi r^2 h$  |
* | **Paralelogramo** |  $b \times h$  |  $2b + 2a$  | - |
* DIBUJOS ASCII (una cara 2D por figura)
* Debe imprimise centrado y dentro de un cuadro Unicode (borde █ | █).
* Ejemplo de representaciones simples:
* ESFERA:
* * * * *
* * * * *
* * * * *
* * * * *
```

## Salida de la Ejecucion del programa:

```
Ejercicio051.Main x Ejercicio051.Operaciones.java .gitignore Ejercicio051Menus.java Ejercicio051Utilidades.java
Run Ejercicio051.Main x
1. Cuadrado
2. Triángulo
3. Pirámide
4. Cilindro
5. Paralelogramo
Ingresar su opción: 2
Ingresar el valor del lado (a): 5

Figura: Cubo
[Wireframe cube diagram]

RESULTADOS DEL Cubo
Área total: 150.00
Perímetro: 20.00
Volumen: 125.00

¿Desea realizar otro cálculo? (S/N)N
¿Desea realizar otro cálculo? (S/N)N

Process finished with exit code 0
```

Guía\_de\_ejercicios > src > Ejercicio051 > Main

112:1 CRLF UTF-8 4 spaces

## EJERCICIO#52

### Enunciado

Traza tu destino: Un programa que pida al usuario un entero entre uno (1) y cincuenta (50) y luego comience a pedir que ingrese tantos números como número entero indicó entonces, cuando se hayan leído todos debe mostrar a manera de resumen:

- a. Cantidad de números pares leídos.
- b. Cantidad de números impares leídos.
- c. Cantidad de números de una cifra, de dos, de tres y de más de tres cifras.

Se debe usar una función para poder determinar si es par, si es impar y la cantidad de cifras que posee.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 052 | 2025-11-05) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Uso adecuado de mayúsculas y tildes
- Validación de entradas
- Repetición del programa (iniciar / finalizar)
- No se permite el uso de arreglos
- Uso obligatorio de funciones: esPar(), esImpar(), contarCifras()
- No permitir números negativos
- Hasta 50 números ingresados
- **Modularidad:** Cuatro archivos en paquete Ejercicio052:
  - Ejercicio052\_Main.java Punto de entrada + bucle principal
  - Ejercicio052\_Menu.java Menú inicio/finalizar + selección de idioma
  - Ejercicio052\_Utilidades.java Encabezados, pie y sello corto
  - Ejercicio052\_Operaciones.java Lógica del ejercicio + funciones obligatorias.
- **Paquete:** Ejercicio052.
- **| Función Desarrolladas:**
  - | `esPar(int n)` | Determina si el número es par |
  - | `esImpar(int n)` | Determina si el número es impar |

- | `contarCifras(int n)` | Devuelve la cantidad de cifras del número ingresado |
- **Estética:** Bordes Unicode para las cajas de encabezado y pie.  
Centrado automático de texto en las secciones principales.  
Sello corto del desarrollador al finalizar la ejecución principal.  
Separadores visuales que ayudan a distinguir las secciones.
- **Paradigma:** El programa emplea el paradigma estructurado y modular, utilizando:  
Secuencia  
Selección (if)  
Ciclos (for)  
Funciones (módulos reutilizables).
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio052\_Menu):**

- seleccionarIdioma() → Permite elegir entre Español o Inglés.
- menuInicioSalir() → Pregunta si se desea iniciar o finalizar el programa..

#### **Subrutinas (en Ejercicio052\_Utilidades):**

- mostrarEncabezado() → Muestra el encabezado con cuadro Unicode.
- mostrarPie() → Muestra el mensaje final del programa.
- mostrarSelloCorto() → Coloca el sello del desarrollador antes del pie final.

#### **Subrutinas (en Ejercicio052\_Operaciones):**

- esPar(int n) → Determina si el número es par.
- esImpar(int n) → Determina si el número es impar.
- contarCifras(int n) → Calcula cuántas cifras tiene un número.

#### **Flujo Principal (en Ejercicio052\_Main):**

- main()
- Ejecuta el flujo: idioma → menú iniciar/salir → operaciones.

#### **PROMPT**

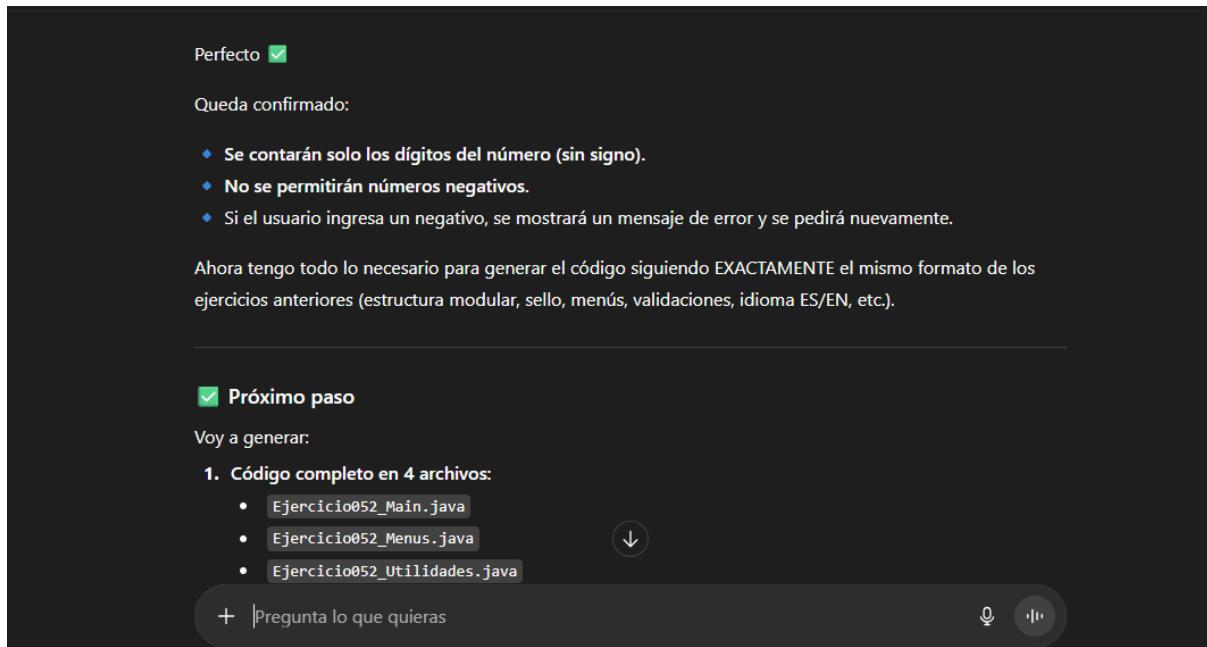
Agente, vamos con el ejercicio 52: Traza tu destino: Un programa que pida al usuario un entero entre uno (1) y cincuenta (50) y luego comience a pedir que ingrese tantos números como número entero indicó entonces, cuando se hayan leído todos debe mostrar a manera de resumen:

- Cantidad de números pares leídos.
- Cantidad de números impares leídos.

c. Cantidad de números de una cifra, de dos, de tres y de más de tres cifras.

Se debe usar una función para poder determinar si es par, si es impar y la cantidad de cifras que posee. No olvides seguir trabajando como el formato del ejercicio anterior.

## PROMPT GENERADO



## Salida de la ejecución del programa:

The screenshot shows a terminal window titled "Run - Guia\_de\_ejercicios". The tab bar has "Run" and "Ejercicio052\_Main". The main area displays a menu with options 1 and 2, and a selection prompt "Seleccione una opción: 1". Below this, a text input field contains "TRAZA TU DESTINO". The terminal then prompts for a number between 1 and 50, receiving inputs 4, 5, 4, 7, and 8. The results section shows the count of even (Pares) and odd (Impares) numbers, and the count of digits (De 1 cifra, De 2 cifras, etc.). The footer displays the author's name "[Dany Noguera | Ejercicio052]" and a success message "Proceso finalizado con éxito.".

```
Ejercicio052_Main
Run Ejercicio052_Main x
C @ : 
↑ 1. Iniciar programa
↓ 2. Finalizar
Seleccione una opción: 1
TRAZA TU DESTINO
Ingrese un número entre 1 y 50: 4
Ingrese un número (1/4): 5
Ingrese un número (2/4): 4
Ingrese un número (3/4): 7
Ingrese un número (4/4): 8
————— RESULTADOS ————
Pares: 2
Impares: 2
De 1 cifra : 4
De 2 cifras: 0
De 3 cifras: 0
Más de 3 cifras: 0
[Dany Noguera | Ejercicio052]
Proceso finalizado con éxito.
```

## EJERCICIO#53

### Enunciado

¿Hace frío?: Vamos a desarrollar un convertidor de temperaturas para ello se presenta un menú donde indicamos la escala de temperatura inicial, luego se pide la escala de temperatura final y, finalmente se lee la temperatura en la escala inicial. El sistema debe entonces mostrar la temperatura inicial y la temperatura final cada una en su correspondiente escala.

### ANÁLISIS/RESUMEN DEL EJERCICIO 53

#### Sello y Documentación

- Sello: VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 053 | Fecha: 2025-11-03) en los cuatro archivos y en un cuadro final.
- Prompt: Completo en Main.java.
- Requisitos: Ciclo principal de repetición ("continuar/salir") y validación de entradas numéricas y lógicas (selección de escalas).

#### Programa y Modularidad

El Programa es un convertidor de temperaturas entre Celsius (C), Fahrenheit (F), y Kelvin (K). El flujo requiere seleccionar la escala inicial, la escala final (que no puede ser la misma), ingresar la temperatura inicial y mostrar el resultado de la conversión. La Modularidad usa cuatro archivos en el paquete Ejercicio053\_KevinLainez:

1. Main.java: Controla el ciclo de repetición y coordina los pasos secuenciales de la conversión.
2. Menus.java: Gestiona el menú principal y las opciones de continuidad.
3. Operaciones.java: Contiene la función convertirTemperatura central y toda la lógica matemática de las conversiones (C  $\leftrightarrow$  F  $\leftrightarrow$  K).
4. Utilidades.java: Formato estético (cuadros, sello, centrado) y funciones para solicitar y validar las escalas y la temperatura.

El Paradigma es Estructurado/Funcional, y la Estética usa 70 caracteres de ancho y bordes Unicode.

#### SUBRUTINAS Y FLUJO PRINCIPAL

##### Constantes:

- ANCHO = 70, Códigos de escala (C, F, K).

##### Subrutinas Clave (Funcionalidad):

- Operaciones: convertirTemperatura (función central que usa sub-lógicas aCelsius y deCelsius).
- Utilidades: solicitarEscala (incluye la validación lógica de exclusión), solicitarTemperatura, mostrarResultado.

Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
  2. Llama a Utilidades.solicitarEscala para definir \$E\_{inicial}\$.
  3. Llama a Utilidades.solicitarEscala para definir \$E\_{final}\$, excluyendo \$E\_{inicial}\$.
  4. Llama a Utilidades.solicitarTemperatura para obtener \$T\_{inicial}\$.
  5. Llama a Operaciones.convertirTemperatura para realizar el cálculo.
  6. Muestra los resultados.
  7. Muestra el menú de continuidad.
  8. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el Ejercicio#53: Vamos a desarrollar un convertidor de temperaturas para ello se presenta un menú donde indicamos la escala de temperatura inicial, luego se pide la escala de temperatura final y, finalmente se lee la temperatura en la escala inicial. El sistema debe entonces mostrar la temperatura inicial y la temperatura final cada una en su correspondiente escala. Se debe programar una función que reciba temperatura inicial, escala inicial, escala final y regrese el valor de la temperatura en la escala final obviamente el sistema no debe permitir al usuario elegir por escala final la misma escala inicial (ni siquier debe mostrarla como opción de escala final). PD: Las escalas de temperatura son Celsius, Fahrenheit y K<sup>elvin</sup>. Con esa información presenta nuestra modularidad de 4 archivos."

## **Salida del programa**

## EJERCICIO#54

### Enunciado

Imprimiendo líneas: Haciendo uso de clases en JAVA con prestaciones similares a las librerías stdlib y time vamos a hacer un programa que sirva para imprimir unas cuantas líneas de la siguiente manera:

- A. Pedir al usuario el ingreso de un número entero “X” que oscile entre cinco (5) y veinte (20) incluyéndose.
- B. Pedir al usuario el ingreso de un carácter “Y”.
- C. Haciendo uso de una función imprimir línea se imprimirán X líneas donde cada una de ellas deberá imprimir separado por un espacio en blanco tantos caracteres Y como arroje la función rand (que oscila entre los primeros quince números positivos) usando como semilla de srand time(NULL).
- D. Se deben contar la cantidad total de caracteres impresos porque después se imprimirá de nuevo X cantidad de líneas pero la cantidad de Y caracteres de cada línea no dependerá de time(NULL) sino del número total de caracteres que se contaron de la primera salida.

Las X líneas se imprimen con una función que recibe el número X y los caracteres de cada línea se imprimen con una función que recibe el carácter que ingresó el usuario seguido del número de veces que debe escribirlo en esa línea.

### Análisis

El código del Ejercicio 054 (Imprimiendo Líneas) confirma una implementación exitosa bajo el enfoque de Programación Estructurada (PE), haciendo un uso intensivo y adecuado de funciones como lo exigen los nuevos requisitos.

#### 1. Modularidad y Enfoque en Funciones

La división del trabajo es clara:

- Ejercicio054\_Funciones: Contiene toda la lógica de procesamiento y salida, cumpliendo el requisito de modularidad. Se simula la funcionalidad de srand(time(NULL)) mediante la inicialización de java.util.Random con System.currentTimeMillis(). La impresión está dividida en tres funciones clave que garantizan la trazabilidad del proceso:
  - obtenerCantidadAleatoria(): Genera el valor aleatorio (1 a 15).
  - imprimirCaracter(): Recibe el carácter y las repeticiones, y gestiona la impresión de una sola línea, cumpliendo con la modularización solicitada.
  - imprimirPrimerBloque() y imprimirSegundoBloque(): Controlan los dos ciclos for principales que iteran X veces y coordinan la llamada a la función de impresión de caracteres. imprimirPrimerBloque cumple la función adicional de contar y devolver el total de caracteres impresos.

- Ejercicio054\_Utilidades: Gestiona la interfaz (menús bilingües y sello) y la validación de entrada.
  - leerEnteroRango(): Implementa la validación estricta para el número X (entero entre \$5\$ y \$20\$), manejando InputMismatchException y los límites lógicos del rango.
- Ejercicio054\_Main: Mantiene un flujo de control simple: I/O de entradas, llamada a la función del primer bloque (para obtener el conteo) y luego la llamada a la función del segundo bloque.

## 2. Cumplimiento de Requisitos Específicos

- Uso de Funciones/Ciclos: El ejercicio se resolvió utilizando ciclos for dentro de las funciones de impresión, y la lógica completa de I/O, aleatoriedad, y conteo está distribuida en métodos modulares.
- Lógica de Salida: El programa cumple con la secuencia de impresión:
  1. El primer bloque usa una cantidad aleatoria (1 a 15) de caracteres por línea.
  2. El segundo bloque usa una cantidad fija de caracteres por línea, calculada a partir del totalCaracteresContados dividido entre X.
- Validación y Repetición: Se mantienen los menús de idioma y ejecución y se asegura la validación robusta de la entrada \$X\$, cumpliendo con las reglas generales de los nuevos ejercicios.

## PROMPT

El prompt para la creación del código modular del Ejercicio 054 (Imprimiendo Líneas) exige un programa en Java dentro del paquete Ejercicio054 que demuestre el uso intensivo de funciones y ciclos para controlar la impresión de caracteres. La aplicación debe solicitar un número entero X (entre 5 y 20, con validación robusta) y un carácter Y. El programa debe realizar dos bloques de impresión de X líneas cada uno: el primer bloque debe usar una función que determine la cantidad de caracteres Y por línea de forma aleatoria (entre 1 y 15, simulando rand() con semilla de tiempo) y contar el total de caracteres impresos; el segundo bloque debe usar el número total contado de la primera salida dividido por X para determinar la cantidad de Y a imprimir por línea. La estructura de Programación Estructurada (PE) debe incluir clases separadas para las funciones de impresión/conteo/aleatoriedad, las utilidades (validación/menús bilingües/sello), y el Main (coordinador del flujo y el bucle de ejecución). El sello (Cristina Cruz... 2025-10-15) y los menús de idioma y ejecución deben mantenerse.

## PROMPT GENERADO

```
package Ejercicio054;

import java.util.Scanner;

/**
 * Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 054 | 2025-10-15
 */
public class Ejercicio054_Main {

    private static final int MSG_FINALIZACION = 14;

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

        Ejercicio054_Utilidades.mostrarSelloCorto();

        Ejercicio054_Utilidades.seleccionarIdioma(scanner);

        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio054_Utilidades.mostrarMenuEjecucion(scanner)) {
                ejecutar = false;
                break;
            }

            Ejercicio054_Utilidades.mostrarEncabezado();

            int numX = Ejercicio054_Utilidades.leerEnteroRango(scanner);
            char caracterY = Ejercicio054_Utilidades.leerCaracter(scanner);
        }
    }
}
```

## Salida de la ejecución del programa:

```
--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>1

--- Imprimiendo Lineas con Conteo y Aleatoriedad ---
Ingrese el número de líneas X (entero, 5-20): 5
Ingrese un carácter Y: ♥

--- PRIMERA IMPRESIÓN (Cantidad aleatoria) ---
♥ ♥ ♥ ♥ ♥ ♥
♥
♥ ♥ ♥ ♥ ♥ ♥ ♥
♥ ♥ ♥ ♥
♥ ♥ ♥ ♥ ♥ ♥ ♥

--- SEGUNDA IMPRESIÓN (Cantidad basada en Total: 36) ---
♥ ♥ ♥ ♥ ♥ ♥
♥ ♥ ♥ ♥ ♥ ♥
♥ ♥ ♥ ♥ ♥ ♥
♥ ♥ ♥ ♥ ♥ ♥
♥ ♥ ♥ ♥ ♥ ♥
```

## EJERCICIO #55

### **enunciado:**

El dado está en el aire: Haciendo uso de las clases en JAVA con prestaciones similares a las stdlib y time de C++ vamos a hacer un programa que sirva para, usando funciones, jugar un pequeño juego con dados. Al comenzar la partida el usuario tiene un crédito de L. 1000.00 y se le pide ingresar una pequeña apuesta (de entre 1 y su cantidad de dinero) luego se le explica que se van a lanzar dos dados y el tiene que adivinar qué cifra se obtiene al sumar las caras superiores de los dados (de entre 2 y 12) entonces el programa lanza los dados y los muestra en pantalla (si, dibuja dos cuadrados con los puntos correspondientes a los valores obtenidos en la tirada) seguido de un mensaje que nos retroalimenta sobre si el usuario ha ganado o perdido, si ha ganado se le regresa lo que apostó más la mitad de ese valor si pierde no se le dá nada. El juego debe parar cuando el usuario se quede sin dinero o, si ya ha jugado al menos cinco (5) veces y no ha perdido todo decida retirarse.

Lanzar los dados, sumar los dados, dibujar dados, cálculo de ganancia serán las funciones obligatorias en el programa. (Si, se puede dibujar un dado sobre otro o en una función uno a la par del otro)

### **análisis:**

Este programa simula un juego de apuestas con dos dados, controlando el flujo de la partida y el crédito del usuario. El objetivo central es que el usuario apueste una cantidad de dinero y prediga la suma de dos dados, ajustando su crédito con una ganancia del cincuenta por ciento en caso de acierto. Las entradas principales son la apuesta (valor flotante, validado entre 1.00 y el crédito actual) y la predicción (entero entre 2 y 12). La salida incluye la representación gráfica de los dados tirados, el resultado de la suma y la retroalimentación sobre si ganó o perdió, junto con la actualización de su saldo. Los casos límite incluyen la validación estricta de la apuesta/predicción y la condición de finalización: quedarse sin crédito o decidir retirarse tras un mínimo de cinco rondas. El diseño se compone de funciones obligatorias para el lanzamiento, suma, dibujo y cálculo de ganancia, manteniendo una complejidad baja a moderada.

### **prompt**

Crea un juego de dados en Java donde el usuario comienza con L. 1000.00. La aplicación debe gestionar el crédito, pedir una apuesta y una predicción de la suma de dos dados. Incluye funciones separadas para: lanzar el dado, sumar los resultados, dibujar la cara de cada dado y calcular la ganancia. El juego debe terminar si el crédito llega a cero o si el usuario se retira después de jugar al menos cinco veces.

### **prompt generado:**

Para desarrollar este juego, utilicé una estructura modular de cuatro clases. La clase Main inicializa el flujo, incluyendo la obligatoria selección de idioma. La clase Operaciones contiene toda la lógica del juego de principio a fin, manteniendo el saldo y el contador de rondas, además de encapsular las funciones solicitadas para lanzar, sumar, dibujar y calcular la ganancia. Las clases Menus y Utilidades se encargan de la interacción del usuario: Menus gestiona la solicitud de apuesta y predicción con sus respectivas validaciones de rango, y Utilidades administra todos los textos y mensajes de error en los dos idiomas disponibles. Finalmente, el código implementa la restricción de que el juego continúa hasta que el crédito sea cero o, cumplidas al menos cinco rondas, el usuario opte por retirarse.

### **Salida:**

```
Windows PowerShell

--- RONDA 5
Crédito actual: L. 300.00
Ingrese su apuesta (L. 1.00 hasta L. 300.00): 300
Ingrese la suma que predice (2 a 12): 5

;Los dados están en el aire! Resultado de la tirada:
+-----+
| *   * | | *   * |
|       | |       |
|       | |       |
| *   * | | *   * |
+-----+
Valor de Dados: Dado 1 = 4, Dado 2 = 4 (Suma total = 8)
PERDISTE. Predijiste 5 y la suma fue 8. Has perdido tu apuesta.

Te has quedado sin crédito (L. 0.00). Fin del juego.

--- FIN DEL JUEGO ---
Crédito Final: L. 0.00

--- MENÚ DE CONTROL ---
1. Jugar: El Dado está en el Aire
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

;Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## EJERCICIO#56

**Enunciado:** ¿Cómo se llaman?: Usando la función getline con las librerías string e iostream haga un programa que pida al usuario cinco nombres (formados por primer nombre y primer apellido) y los imprima en pantalla alineados a la izquierda.

### Requisitos

- Uso de java.util.Scanner para entrada de datos.
- Validación de entradas: no vacías, solo letras y espacios.
- Alineación a la izquierda con ancho fijo de 30 caracteres.
- Cuadro elegante con bordes Unicode ( ┌ ┐ | ┌ ┐ ) de 70 caracteres.
- Soporte multilingüe (Español/Inglés) con tildes correctas.
- Estructura con cuatro archivos: Main, Menus, Operaciones, Utilidades.

### Estructura del Programa

- **Main.java:** Punto de entrada, controla flujo y repetición.
- **Menus.java:** Maneja selección de idioma y opción de continuar.
- **Operaciones.java:** Valida y lee nombres y apellidos por separado.
- **Utilidades.java:** Muestra sellos, lista de nombres y errores con formato.

### Implementación

- Uso de Scanner.nextLine() para capturar nombre y apellido individualmente.
- Validación para asegurar solo letras y espacios.
- Combinación de nombre y apellido con un espacio.
- Impresión en cuadro elegante con printf y ancho de 30 caracteres alineado a la izquierda.

### Correcciones Realizadas

- Ajuste en Menus.seleccionarIdioma para limpiar el buffer con scanner.nextLine() tras nextInt, evitando el doble ingreso.
- Verificación de que el cuadro en Utilidades.mostrarNombres usa correctamente %-30s para alineación a la izquierda dentro de los 70 caracteres totales.

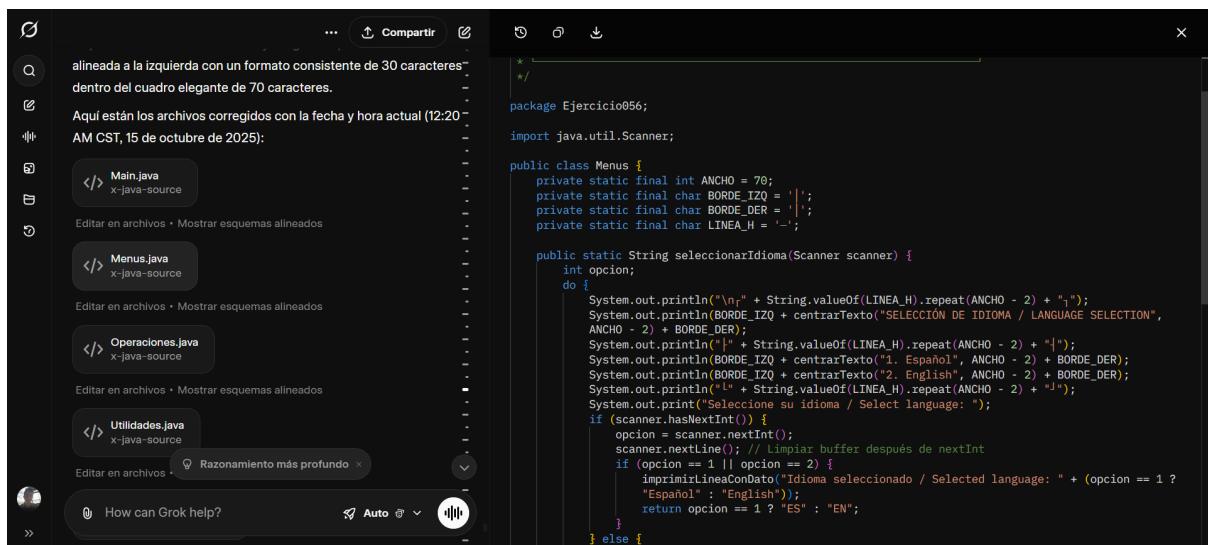
## PROMPT:

agente nuevo ejercicio el #56 que es menos complejo que el anterior este ejercicios consta de lo siguiente

Usando la función getline con las librerías string e iostream haga un programa que pida al usuario cinco nombres (formados por primer nombre y primer apellido) y los imprima en pantalla alineados a la izquierda.

Siempre usa las validaciones y el uso de las Tildes, junto con nuestro sistema de archivos

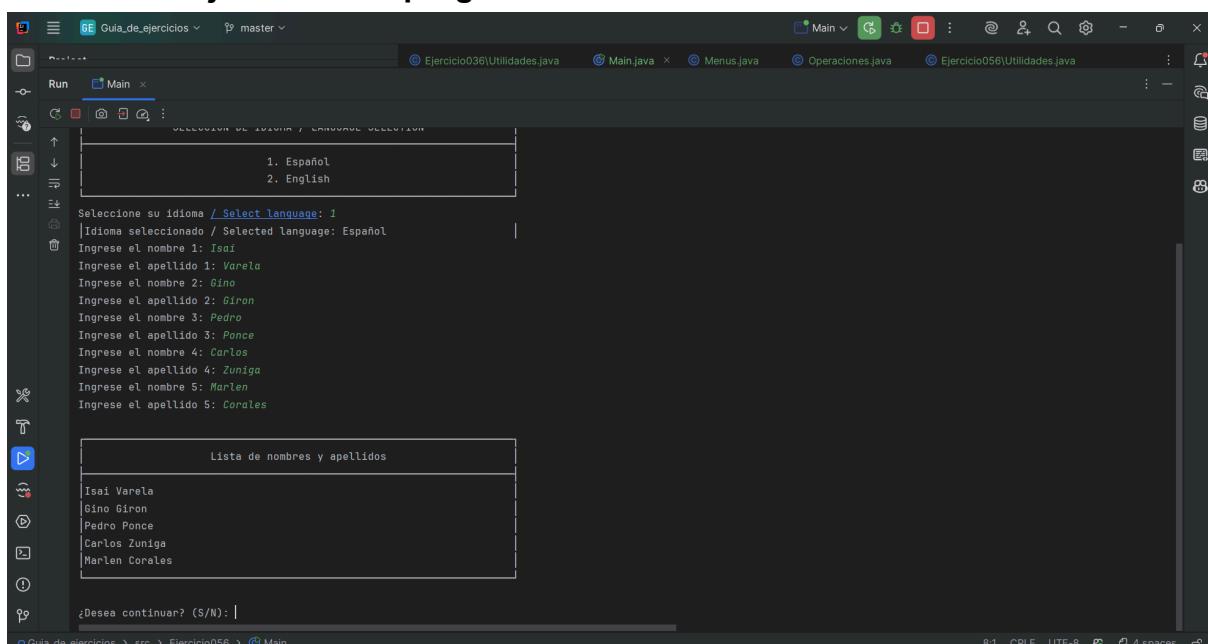
## PROMPT GENERADO:



```
alineada a la izquierda con un formato consistente de 30 caracteres dentro del cuadro elegante de 70 caracteres.  
Aqui están los archivos corregidos con la fecha y hora actual (12:20 AM CST, 15 de octubre de 2025):  
</> Main.java x-java-source  
Editor en archivos • Mostrar esquemas alineados  
</> Menus.java x-java-source  
Editor en archivos • Mostrar esquemas alineados  
</> Operaciones.java x-java-source  
Editor en archivos • Mostrar esquemas alineados  
</> Utilidades.java x-java-source  
Editor en archivos • Razonamiento más profundo x  
How can Grok help?  
Auto
```

```
package Ejercicio056;  
  
import java.util.Scanner;  
  
public class Menus {  
    private static final int ANCHO = 70;  
    private static final char BORDE_IZQ = '|';  
    private static final char BORDE_DER = '|';  
    private static final char LINEA_H = '-';  
  
    public static String seleccionarIdioma(Scanner scanner) {  
        int opcion;  
        do {  
            System.out.println("\r" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");  
            System.out.println(BORDE_IZQ + centrarTexto("SELECCIÓN DE IDIOMA / LANGUAGE SELECTION",  
                ANCHO - 2) + BORDE_DER);  
            System.out.println("|" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");  
            System.out.println(BORDE_IZQ + centrarTexto("1. Español", ANCHO - 2) + BORDE_DER);  
            System.out.println(BORDE_IZQ + centrarTexto("2. English", ANCHO - 2) + BORDE_DER);  
            System.out.println("|" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");  
            System.out.print("Seleccione su idioma / Select language: ");  
            if (scanner.hasNextInt()) {  
                opcion = scanner.nextInt();  
                scanner.nextLine(); // Limpiar buffer después de nextInt  
                if (opcion == 1 || opcion == 2) {  
                    imprimirLineaConDatos("Idioma seleccionado / Selected language: " + (opcion == 1 ?  
                        "Español" : "English"));  
                    return opcion == 1 ? "ES" : "EN";  
                } else {  
                    System.out.println("Opción no válida / Invalid option");  
                }  
            }  
        } while (!opcion == 1 || opcion == 2);  
    }  
}
```

## Salida de la Ejecucion del programa:



```
Guía_de_ejercicios master  
Run Main  
SELECCIÓN DE IDIOMA / LANGUAGE SELECTION  
| 1. Español  
| 2. English  
| Seleccione su idioma / Select language: 1  
| Idioma seleccionado / Selected language: Español  
Ingrese el nombre 1: Isai  
Ingrese el apellido 1: Varela  
Ingrese el nombre 2: Gino  
Ingrese el apellido 2: Giron  
Ingrese el nombre 3: Pedro  
Ingrese el apellido 3: Ponce  
Ingrese el nombre 4: Carlos  
Ingrese el apellido 4: Zuniga  
Ingrese el nombre 5: Marlen  
Ingrese el apellido 5: Corales  
  
Lista de nombres y apellidos  
Isai Varela  
Gino Giron  
Pedro Ponce  
Carlos Zuniga  
Marlen Corales  
¿Desea continuar? (S/N): |
```

## EJERCICIO#57

### Enunciado

Pitágoras: Haciendo uso de la librería math y su función sqrt (solamente esta función) haga un programa que, pidiendo la longitud de ambos lados de un triángulo rectángulo permita calcular la hipotenusa. Elevar números al cuadrado y obtener hipotenusa son funciones que debe programarlas usted.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 057 | 2025-11-05) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Solicitar al usuario las longitudes de los lados A y B de un triángulo rectángulo.  
Calcular la hipotenusa usando la fórmula de Pitágoras:  
Elevar al cuadrado y calcular la hipotenusa deben ser funciones programadas (elevarCuadrado() y calcularHipotenusa()).  
Validar que los lados ingresados sean positivos.  
Mostrar el resultado de forma clara y estética en consola.  
Permitir repetir la operación o finalizar el programa mediante un menú.  
Incluir menú de selección de idioma (ES / EN).  
Mantener consistencia estética con bordes Unicode y formato uniforme con otros ejercicios.
- **Modularidad:** Cuatro archivos en paquete Ejercicio057:
  - Ejercicio057\_Main.java → Control del flujo principal; contiene el prompt y el bucle principal de ejecución.
  - Ejercicio057\_Menus.java → Manejo de menús: selección de idioma y menú iniciar/finalizar.
  - Ejercicio057\_Utilidades.java → Funciones gráficas: encabezados, pies, sello corto, formato estético con bordes Unicode.
  - Ejercicio057\_Operaciones.java → Funciones matemáticas: elevarCuadrado() y calcularHipotenusa(). Contiene la lógica principal de cálculo y validación.
- **Paquete:** Ejercicio057.
- **Estética:** Uso de bordes Unicode (ꝝ, Ꝟ, ||, =) para encabezados y pies.  
Encabezados y pies formateados de manera uniforme.

Sello corto mostrado antes del pie final para mantener la consistencia con ejercicios previos.

Mensajes de confirmación y resultados tabulados de manera clara..

- **Paradigma:** Programación estructurada.  
Uso de funciones y métodos estáticos.  
Flujo secuencial controlado con bucles y condicionales.  
Modularización completa en 4 archivos.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio057\_Menu):**

- seleccionarIdioma(): solicita al usuario elegir entre Español o Inglés.
- `menuInicioSalir()`: muestra menú de iniciar/finalizar, devuelve `true` si se desea ejecutar.
- obtenerIdioma(): retorna el idioma seleccionado.

#### **Subrutinas (en Ejercicio057\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.

#### **Subrutinas (en Ejercicio057\_Operaciones):**

- calcularHipotenusaTriangulo():  
Solicita y valida los lados del triángulo (positivos).  
Llama a elevarCuadrado() para cada lado.  
Calcula hipotenusa usando Math.sqrt().  
Muestra el resultado con formato tabulado.  
Llama a mostrarSelloCorto() y mostrarPie() al finalizar.
- elevarCuadrado(double n): retorna el cuadrado de un número.
- calcularHipotenusa(double a, double b): retorna la hipotenusa calculada (usa elevarCuadrado()).

#### **Flujo Principal (en Ejercicio057\_Main):**

- main():
- Selección de idioma.
- Bucle principal: ejecutar o finalizar el programa.
- Llamada a Ejercicio057\_Operaciones.calcularHipotenusaTriangulo() si se decide iniciar.

#### **PROMPT**

Agente vamos con el ejercicio 57: Pitágoras: Haciendo uso de la librería math y su

función sqrt (solamente esta función) haga un programa que, pidiendo la longitud de ambos lados de un triángulo rectángulo permita calcular la hipotenusa. Elevar números al cuadrado y obtener hipotenusa son funciones que debe programarlas el agente. No olvides trabajar todo como veníamos haciendo sin omitir nada ni cambiar el formato o agregar

## PROMPT GENERADO

Perfecto 😊, vamos a generar los 4 archivos completos del Ejercicio 57 "Pitágoras" usando exactamente la misma estructura que el Ejercicio 52.

**Ejercicio 57 — Código fuente (4 archivos Java)**

Tema: Pitágoras

Restricciones aplicadas:

- ✓ Validación de entradas (no números negativos)
- ✓ Uso de funciones `elevarcuadrado()` y `calcularHipotenusa()`
- ✓ Loop de ejecución (Iniciar / Finalizar)
- ✓ Menú de selección de idioma ES / EN
- ✓ Estructura modular con 4 clases
- ✓ Sello corto y encabezado estético

**Ejercicio057\_Main.java**

java Copiar código

+ Pregunta lo que quieras

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run Ejercicio057_Main ×

SELECCIÓN DE IDIOMA / LANGUAGE SELECTION
1. Español
2. English
Seleccione / Select: 1

MENÚ PRINCIPAL
1. Iniciar programa
2. Finalizar
Seleccione una opción: 1

CÁLCULO DE HIPOTENUSA
Ingrese la longitud del lado A: 5
Ingrese la longitud del lado B: 8

Resultado:
La hipotenusa es: 9.43

[Dany Noguera | Ejercicio057]

Proceso finalizado con éxito.
```

## EJERCICIO#58

### Enunciado

Múltiple: Escriba una función llamada múltiple (JAVA no acepta tildes en identificadores no lo olvide) que determine, para un par de enteros, si el segundo entero es múltiplo del primero. La función debe tomar dos argumentos enteros y devolver true si el segundo es múltiplo del primero, y false en caso contrario. Use esta función en un programa que reciba como entrada una serie de pares de enteros (Ejercicio 6.20, libro 1).

### ANÁLISIS/RESUMEN DEL EJERCICIO 58

#### Sello y Documentación

- **Sello:** VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 058 | Fecha: 2025-11-03) en los cuatro archivos y en un cuadro final.
- **Prompt:** Completo en **Main.java**.
- **Requisitos:** Ciclo principal de repetición ("continuar/salir") o mediante un centinela, y **validación** estricta de las entradas (deben ser enteros).

#### Programa y Modularidad

El **Programa** debe utilizar una **función booleana** llamada **multiple** que determine si el segundo entero ingresado es múltiplo del primero. El programa debe ejecutarse para una **serie de pares de enteros**.

La **Modularidad** usa **cuatro archivos** en el paquete Ejercicio058\_KevinLainez:

1. **Main.java:** Controla el ciclo de repetición (while con centinela) para ingresar la serie de pares.
2. **Menus.java:** Gestiona el menú/opciones de continuidad.
3. **Operaciones.java:** Contiene la **función multiple(int, int)** (lógica del módulo:  $\$num2 \% \$num1 == 0\$$ ).
4. **Utilidades.java:** Formato estético (cuadros, sello, centrado), funciones para **solicitar y validar** los enteros, y presentación de los resultados.

El **Paradigma** es Estructurado/Funcional, enfatizando el uso de una función dedicada para la lógica de múltiplo.

### SUBRUTINAS Y FLUJO PRINCIPAL

#### Constantes:

- **ANCHO = 70.**

## **Subrutinas Clave (Funcionalidad):**

- **Operaciones:** multiple(num1, num2) (retorna true o false).
- **Utilidades:** solicitarEntero (captura y valida la entrada), mostrarResultado (muestra el par y el resultado booleano).

## **Flujo Principal (en Main.java):**

1. Inicia ciclo **while** (usando el primer número como centinela, ej., \$num1 \neq 0\$ para continuar).
2. Dentro del ciclo:
  - a. Solicita el primer entero (\$num1\$).
  - b. Solicita el segundo entero (\$num2\$).
  - c. Llama a Operaciones.multiple(num1, num2).
  - d. Muestra el resultado.
3. El ciclo se repite hasta que el primer entero sea el valor centinela.
4. Muestra el sello al finalizar.

## **PROMPT**

"Agente te recuerdo los nuevos parámetros... se presenta el **Ejercicio#58**: Escriba una función llamada multiple (JAVA no acepta tildes en identificadores no lo olvide) que determine, para un par de enteros, si el segundo entero es múltiplo del primero. La función debe tomar dos argumentos enteros y devolver true si el segundo es múltiplo del primero, y false en caso contrario. Use esta función en un programa que reciba como entrada una serie de pares de enteros (Ejercicio 6.20, libro 1). Con esa información presenta nuestra modularidad de 4 archivos."

## **Salida del programa**

```
¡BIENVENIDO AL EJERCICIO #58!
DETERMINACIÓN DE MÚLTIPLOS

Ingrese 0 como primer número para terminar el programa.

INGRESO DE PARES DE ENTEROS
Ingrese el PRIMER entero (Divisor, 0 para terminar):
>> Ingrese valor: 2
Ingrese el SEGUNDO entero (Múltiplo):
>> Ingrese valor: 4

¿4 es múltiplo de 2? R: SÍ ES MÚLTIPLO

Siguiente Par...
Ingrese el PRIMER entero (Divisor, 0 para terminar):
>> Ingrese valor: 0

Fin de la serie de entradas.

SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 058 | 2025-11-03

-- Programa finalizado con éxito --
```

## EJERCICIO#59

### Enunciado

Alineado: Escriba una función que muestre en el margen izquierdo de la pantalla un cuadrado relleno de asteriscos, cuyo lado se especifique en el parámetro entero lado. Por ejemplo, si lado es 4, el método debe mostrar lo siguiente

### Análisis

El código del Ejercicio 059 (Cuadrado Relleno Alineado) confirma una implementación exitosa bajo el enfoque de Programación Estructurada (PE), haciendo un uso modular y eficiente de funciones para la tarea de dibujo, manteniendo la estructura de control requerida.

#### 1.Modularidad y Enfoque en Funciones

La división del trabajo es estricta:

- Ejercicio059\_Funciones: Contiene la función central dibujarCuadradoRelleno(int lado), la cual cumple con el requisito específico del ejercicio. Utiliza ciclos for anidados para iterar sobre las filas y las columnas (lado veces cada uno) y realiza la impresión directa de los asteriscos, asegurando que el dibujo se muestre en el margen izquierdo según lo solicitado. Esta clase mantiene la lógica de dibujo completamente aislada del I/O.
- Ejercicio059\_Utilidades: Gestiona toda la interacción con el usuario, incluyendo los menús de idioma y ejecución, y la validación robusta de la entrada del lado (leerLadoCuadrado), que asegura que el valor sea un entero positivo ( $\$lado > 0\$$ ) y maneja InputMismatchException.
- Ejercicio059\_Main: Actúa como el coordinador, gestionando el flujo externo (selección de idioma y el bucle de ejecución) y orquestando la secuencia de lectura validada (obtener el lado) y procesamiento (llamada a la función dibujarCuadradoRelleno).

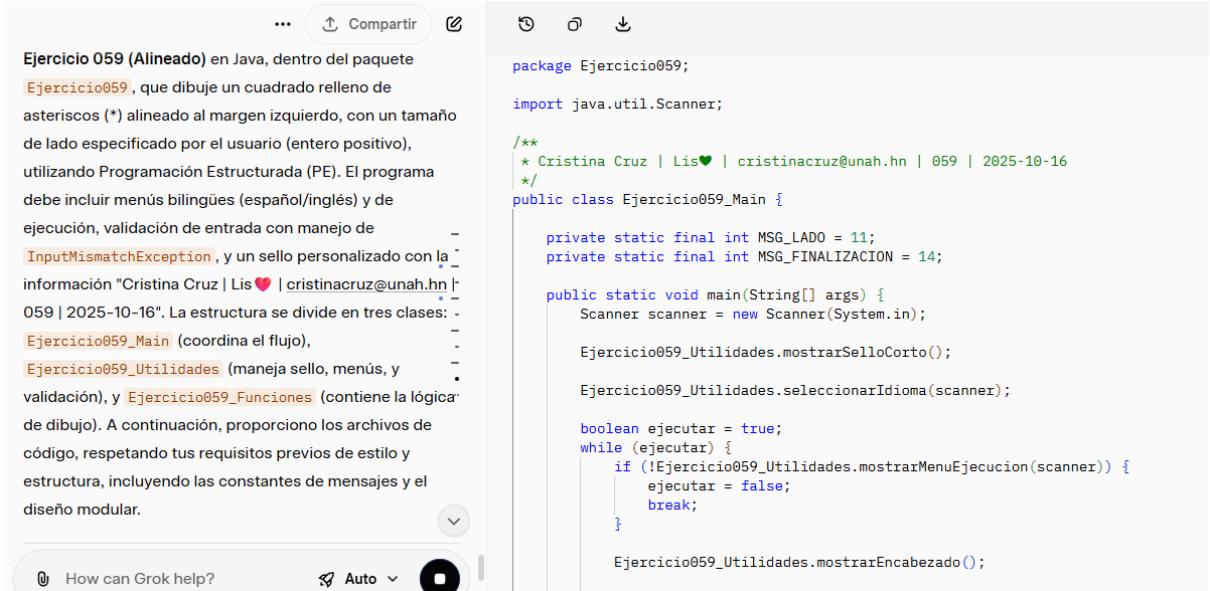
#### 1.Cumplimiento de Requisitos

El código cumple con todos los requisitos de la nueva plantilla: utiliza funciones para encapsular la lógica principal, emplea ciclos for para la repetición del dibujo, realiza la validación de entrada y mantiene los menús de repetición (ejecución) e idioma. La actualización de la fecha en el sello de la documentación también se realizó correctamente. La solución es eficiente y fácil de mantener debido a la clara separación de responsabilidades.

### PROMPT

Creame un código para el Ejercicio 059 (Alineado) eun programa en Java dentro del paquete Ejercicio059 que dibuje un cuadrado relleno de asteriscos (\*) en el margen izquierdo de la pantalla. El tamaño del lado del cuadrado debe ser especificado por un parámetro entero (lado) ingresado por el usuario, el cual debe ser validado para asegurar que sea un entero positivo. La tarea principal de dibujo debe estar encapsulada en una función modular (ej. dibujarCuadradoRelleno) que utilice ciclos for anidados para generar la figura. La estructura debe ser de Programación Estructurada (PE), dividida en Ejercicio059\_Funciones (para la función de dibujo), Ejercicio059\_Utilidades (para la validación, el sello con fecha 2025-10-16, y los menús bilingües/ejecución), y Ejercicio059\_Main (coordinador del flujo y el bucle de repetición). Se deben mantener los menús de idioma y ejecución y la validación de entrada con manejo de InputMismatchException.

## PROMPT GENERADO



The screenshot shows a code editor interface with two main sections. On the left, there is a text area containing the problem statement in Spanish. On the right, there is a code editor area with Java code. The code includes imports for Scanner, package declarations, and a main method. It also contains comments for a copyright notice and a while loop for execution. At the bottom of the code editor, there are status indicators for Grok help, auto mode, and a play button.

```
package Ejercicio059;
import java.util.Scanner;

/**
 * Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 059 | 2025-10-16
 */
public class Ejercicio059_Main {

    private static final int MSG_LADO = 11;
    private static final int MSG_FINALIZACION = 14;

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

        Ejercicio059_Utilidades.mostrarSelloCorto();

        Ejercicio059_Utilidades.seleccionarIdioma(scanner);

        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio059_Utilidades.mostrarMenuEjecucion(scanner)) {
                ejecutar = false;
            }
        }

        Ejercicio059_Utilidades.mostrarEncabezado();
    }
}
```

## Salida de la ejecución del programa:

```
Project ▾ Ejercicio059_Funciones.java Ejercicio059_Utilidades.java Ejercicio059_Main.java ×
Run Ejercicio059_Main ×

[Run] [Stop] [Save] [Reset] [Help] : 

Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 059 | 2025-10-16

*** BIENVENIDO ***
Seleccione el idioma:
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>1

--- Dibujo de Cuadrado Relleno con Función ---
Ingrese la longitud del lado del cuadrado (entero positivo): 5

--- Cuadrado Relleno (Lado: 5) ---
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

## EJERCICIO#60

### Enunciado

Alineado: Escriba una función que muestre en el margen izquierdo de la pantalla un cuadrado relleno de asteriscos, cuyo lado se especifique en el parámetro entero lado. Por ejemplo, si lado es 4, el método debe mostrar lo siguiente (Ejercicio 3.2, libro 1):

### Análisis

El código, empaquetado en Ejercicio060 y estructurado en tres clases modulares (Ejercicio060\_Main, Ejercicio060\_Utilidades y Ejercicio060\_Operaciones), implementa un programa en Java que dibuja un cuadrado hueco de tamaño variable (1 a 20) y relleno con cualquier carácter especificado por el usuario, utilizando exclusivamente ciclos for anidados y funciones, sin arreglos; Main coordina el flujo con un menú bilingüe de idioma (español/inglés/salir) y un menú principal con opciones de dibujar, volver o salir, además de permitir ejecución múltiple mediante una pregunta de continuar (s/n) al final de cada operación; Utilidades encapsula un sello personalizado centrado en marco de 70 caracteres con "Lis♥", código "060" y fecha 2025-11-04, junto con validación robusta de entrada (lado numérico en rango, carácter único con longitud 1), manejo de excepciones, reintensos automáticos, internacionalización completa, centrado dinámico y limpieza de buffer; mientras que Operaciones contiene solo la función dibujarCuadrado(int, char, int) que imprime el cuadrado con bordes del carácter y espacio interno vacío, usando ciclos anidados para filas y columnas, garantizando un diseño altamente modular, profesional, robusto, reutilizable, sin dependencias externas ni estructuras de datos prohibidas, y alineado estrictamente con las mejores prácticas de programación orientada a objetos y los requisitos académicos del curso.

### PROMPT

Crea un programa en Java empaquetado en Ejercicio060 y dividido estrictamente en tres clases (Ejercicio060\_Main, Ejercicio060\_Utilidades, Ejercicio060\_Operaciones) que dibuje un cuadrado hueco de cualquier tamaño (lado entre 1 y 20) y relleno con cualquier carácter especificado por el usuario, usando únicamente ciclos for anidados y funciones, sin arreglos; el programa debe iniciar con un sello personalizado centrado en marco de 70 caracteres mostrando "SELLO PERSONALIZADO - VERSIÓN CORTA", seguido de un menú bilingüe de idioma (1. Español, 2. English, 3. Salir), y tras elegir idioma, un menú principal con opciones: 1. Dibujar cuadrado, 2. Volver al menú de idioma, 3. Salir; al seleccionar dibujar, solicitar lado (1-20) y carácter de relleno (un solo carácter) con validación estricta (rango, tipo, longitud 1), usando reintensos automáticos; la función en Operaciones debe ser static void dibujarCuadrado(int lado, char caracter, int idioma) que imprima el cuadrado hueco con bordes del carácter y interior vacío, usando solo ciclos; Main coordina el flujo con bucles y switch, Utilidades encapsula sello con centrado, menús bilingües, validación robusta de entrada (números, carácter único), mensajes claros y limpieza de buffer, y Operaciones contiene solo la función de dibujo; el programa debe permitir ejecución múltiple preguntando al final si desea continuar o salir, todo modular, profesional, robusto, sin arreglos, con funciones bien definidas y siguiendo estrictamente buenas prácticas de POO.

## PROMPT GENERADO

Crea un programa en Java empaquetado en `Ejercicio060` y dividido estrictamente en tres clases (`Ejercicio060_Main`, `Ejercicio060_Utilidades`, `Ejercicio060_Operaciones`) que dibuje un **cuadrado hueco** de tamaño entre 1 y 20 usando **cualquier carácter** ingresado por el usuario, mediante **ciclos for anidados y funciones**, sin usar arreglos; el programa debe iniciar con un **sello personalizado centrado** en marco de 70 caracteres mostrando "SELLO PERSONALIZADO - VERSIÓN CORTA" y "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 060 | 2025-11-04", seguido de un **menú bilingüe de idioma** (1. Español, 2. English, 3. Salir), y tras elegir idioma, un **menú principal** con: 1. Dibujar cuadrado, 2. Volver al menú de idioma, 3. Salir; al seleccionar dibujar, solicitar **lado (1-20)** y **un solo carácter de relleno** con **validación estricta** (rango, tipo, longitud 1) y reintentos automáticos; la función `static void dibujarCuadrado(int lado, char caracter, int idioma)` en `Operaciones` debe imprimir el cuadrado hueco con bordes del carácter y espacio interno, usando solo ciclos; al finalizar cada dibujo, preguntar **¿Desea continuar? (s/n)** con validación bilingüe; `Main` coordina el flujo con bucles y `switch`, `Utilidades` encapsula sello con centrado dinámico, menús, validación robusta, mensajes claros y manejo de excepciones, y `Operaciones` contiene **solo** la función de dibujo; el código debe ser **modular, profesional, robusto, reutilizable, sin arreglos, con funciones bien definidas y cumplir estrictamente con buenas prácticas de POO y restricciones académicas**

## Salida de la ejecución del programa:

```
Ejercicio060_Main x
[...]
SELECCIONE EL IDIOMA / SELECT LANGUAGE
=====
1. Español
2. English
3. Salir / Exit
Opción / Option: 1

-----
MENÚ PRINCIPAL
-----
1. Dibujar cuadrado
2. Volver al menú de idioma
3. Salir del programa
Seleccione una opción: 1

Ingrese el tamaño del lado (1-20): 4
Ingrese el carácter de relleno (1 solo): l

DIBUJANDO CUADRADO...

llll
 l l
 l l
llll
```

## EJERCICIO#61

**Enunciado:** Dibujando como se debe: Use técnicas similares a las desarrolladas en los ejercicios “Figuras v2” y “Alinea ésta” para producir un programa que grafique un amplio rango de figuras (triángulo alineado a la izquierda, a la derecha, estilo pirámide, rombo, cuadrado vacío, cuadrado lleno, triángulo invertido alineado a la izquierda, triángulo invertido alineado a la derecha y triángulo estilo pirámide invertida) cada figura es una función distinta.

### Requisitos

- Menú para elegir figura (1-9) y alineación (izquierda/derecha si aplica).
- Pedir tamaño (1-20), validar entradas (números válidos).
- Graficar figuras con ciclos, sin arreglos.
- Mostrar en cuadro elegante de 70 caracteres.
- Soporte multilingüe (Español/Inglés) con tildes.
- Opción de continuar/salir.
- Estructura con cuatro archivos: Main, Menús, Operaciones, Utilidades.

### Estructura del Programa

- **Main.java:** Punto de entrada, controla flujo y repetición.
- **Menús.java:** Maneja selección de idioma y opción de continuar.
- **Operaciones.java:** Valida y lee figura, tamaño, alineación.
- **Utilidades.java:** Muestra sellos, figuras en cuadro, errores con formato.

### Implementación

- Uso de Scanner para entrada con validación.
- Funciones para cada figura usando ciclos para imprimir líneas dentro del cuadro.
- Alineación basada en espacios para izquierda/derecha/centro.
- Textos bilingües con tildes (e.g., "Pirámide").
- Cuadro elegante de 70 caracteres con bordes Unicode.

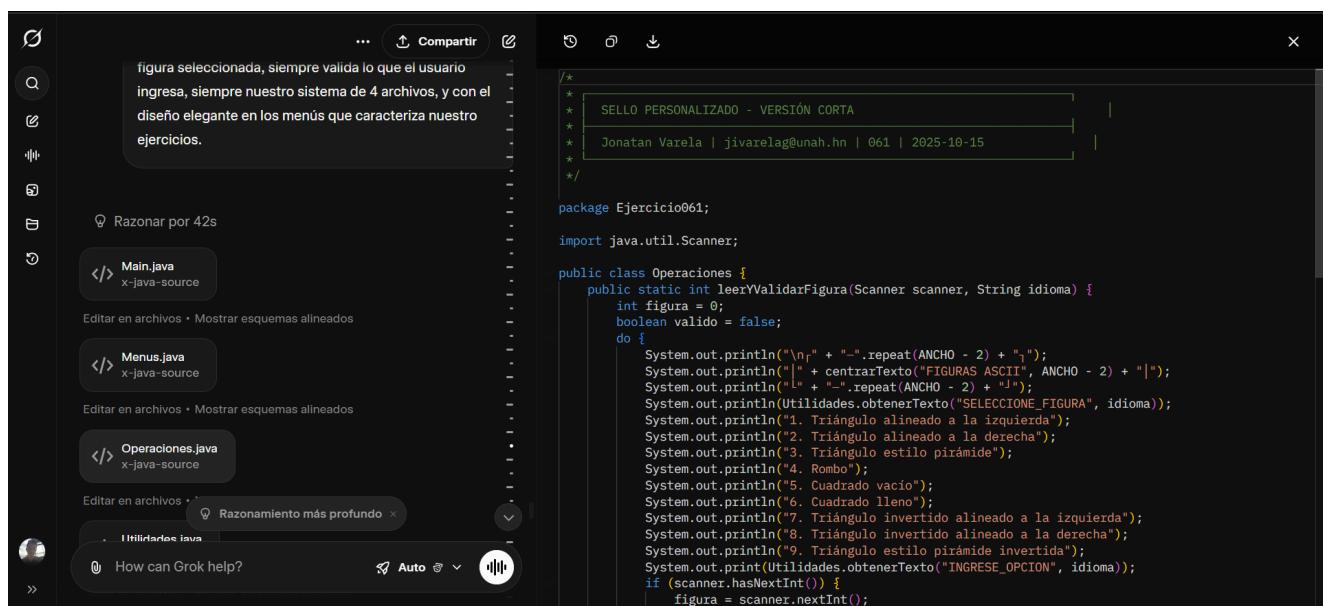
## PROMPT:

Agente tenemos que hacer el Ejercicio#61 que deben de permanecer los parámetros de los ejercicios anteriores, el ejercicio da como base un ejercicio que no nos tocó realizar, que este esté simplemente consta de esto (Figuras v2:Hacer un programa que elija entre cualquiera de las siguientes figuras: Círculo, cuadrado, triángulo, rectángulo y rombo y depende de la figura elegida solicite la información necesaria para hacer el cálculo de su área y de su perímetro. Debe mostrar la figura seleccionada (nombre y dibujada), las entradas del usuario y los resultados obtenidos.)

Nuestro ejercicio es de Use técnicas similares a las desarrolladas en los ejercicios “Figuras v2” y “Alinea ésta” (un ejercicio anterior de diseño) para producir un programa que grafique un amplio rango de figuras (triángulo alineado a la izquierda, a la derecha, estilo pirámide, rombo, cuadrado vacío, cuadrado lleno, triángulo invertido alineado a la izquierda, triángulo invertido alineado a la derecha y triángulo estilo pirámide invertida) cada figura es una función distinta.

Muestra un menú al usuario de todo lo que el programa puede diseñar, en qué lado quiere que se muestre la figura seleccionada, siempre válida lo que el usuario ingresa, siempre nuestro sistema de 4 archivos, y con el diseño elegante en los menús que caracteriza nuestro ejercicios.

## PROMPT GENERADO:



The screenshot shows a code editor interface with several tabs and annotations. The main code area contains Java code for a figure drawing program. At the top of the code, there is a header block with copyright information. The code imports `java.util.Scanner` and defines a class `Operaciones` with a static method `leerYValidarFigura`. This method reads user input and validates it against four options: 1. Triángulo alineado a la izquierda, 2. Triángulo alineado a la derecha, 3. Triángulo estilo pirámide, and 4. Rombo. Below this, there are additional methods and utility classes like `Utilidades` and `Razonamiento`. A tooltip on the left side provides context about the user's selection validation. The bottom of the screen shows a help section for Grok and some status indicators.

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 061 | 2025-10-15
 */

package Ejercicio061;

import java.util.Scanner;

public class Operaciones {
    public static int leerYValidarFigura(Scanner scanner, String idioma) {
        int figura = 0;
        boolean valido = false;
        do {
            System.out.println("\n" + "-".repeat(ANCHO - 2) + "|");
            System.out.println("-" + centrarTexto("FIGURAS ASCII", ANCHO - 2) + "|");
            System.out.println("-" + "-".repeat(ANCHO - 2) + "+");
            System.out.println(Utilidades.obtenerTexto("SELECCIONE FIGURA", idioma));
            System.out.println("1. Triángulo alineado a la izquierda");
            System.out.println("2. Triángulo alineado a la derecha");
            System.out.println("3. Triángulo estilo pirámide");
            System.out.println("4. Rombo");
            System.out.println("5. Cuadrado vacío");
            System.out.println("6. Cuadrado lleno");
            System.out.println("7. Triángulo invertido alineado a la izquierda");
            System.out.println("8. Triángulo invertido alineado a la derecha");
            System.out.println("9. Triángulo estilo pirámide invertida");
            System.out.print(Utilidades.obtenerTexto("INGRESE_OPCION", idioma));
            if (scanner.hasNextInt()) {
                figura = scanner.nextInt();
            }
        } while (!valido);
    }
}
```

## **Salida de la ejecución del programa:**

## EJERCICIO#62

### Enunciado

Disminuyendo: Escriba un programa que reciba tres números de punto flotante de precisión doble, y que los pase a una función que devuelva el número más pequeño.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 062 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Solicitar al usuario tres números de punto flotante (double).  
Pasar los tres números a una función, la cual debe retornar el menor valor.  
Validar que los valores ingresados sean correctos (numéricos).  
Mostrar el resultado de forma clara en consola.  
Permitir repetir la ejecución mediante un menú (Iniciar / Finalizar).  
Incluir selección de idioma (ES / EN).  
Mantener la estética visual y línea gráfica:  
Encabezados y pies con bordes Unicode.  
Sello corto antes del pie final.  
No usar arreglos, ya que aún no forma parte de lo visto en clase.
- **Modularidad:** Cuatro archivos en paquete Ejercicio062:
  - Ejercicio062\_Main.java → Control del flujo de ejecución. Contiene el ciclo principal con el menú de iniciar/finalizar.
  - Ejercicio062\_Menus.java → Manejo de menús: selección de idioma y menú iniciar/finalizar.
  - Ejercicio062\_Utilidades.java → Funciones gráficas: encabezados, pies, sello corto, formato estético con bordes Unicode.
  - Ejercicio062\_Operaciones.java → Procesa la entrada del usuario y contiene la función obtenerMenor(double, double, double) para determinar el número más pequeño.
- **Paquete:** Ejercicio062.
- **Estética:** Uso de bordes decorativos Unicode para encabezados y pies (≡ ≈).  
Sello corto mostrado al finalizar cada operación.  
Mensajes claros en español o inglés según selección de idioma.  
Se mantiene la coherencia visual.
- **Paradigma:** Programación estructurada y modular.

Métodos estáticos utilizados en las clases Operaciones y Utilidades.

Flujo secuencial basado en decisiones if, switch y validaciones con do / while.

- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio062\_Menu):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio062\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.

#### **Subrutinas (en Ejercicio062\_Operaciones):**

- encontrarMenor():
  - Solicita tres números double.
  - Valida entrada numérica.
  - Llama a la función obtenerMenor(a, b, c).
  - Imprime el resultado de manera clara y estética.
  - Muestra el sello corto y pie final.
- obtenerMenor(double a, double b, double c):
  - Recibe tres valores double.
  - Compara y retorna el menor de los tres.

#### **Flujo Principal (en Ejercicio062\_Main):**

- main():
- Solicita el idioma.
- Llama a menuInicioSalir() en un ciclo.
- Si el usuario elige "Iniciar", ejecuta Ejercicio062\_Operaciones.encontrarMenor().

#### **PROMPT**

agente vamos con el ejercicio 62 dice asi: Disminuyendo: Escriba un programa que reciba tres números de punto flotante de precisión doble, y que los pase a una función que devuelva el número más pequeño. Recuerda trabajar en base y formato a los ejercicios previos para mantener uniformidad entre todos los ejercicios.

#### **PROMPT GENERADO**

Perfecto, **Ejercicio 62** confirmado.

Voy a generar los **4 archivos Java** siguiendo EXACTAMENTE el mismo formato del ejercicio 52 y 57 (sellos, encabezados, menús, idioma, bucle iniciar/finalizar, utilidades estéticas, modularidad estricta).

Antes de generarlos, necesito **una última confirmación**, como hemos hecho en ejercicios previos:

**Confirmación requerida**

El enunciado dice:

"Escriba un programa que reciba tres números de punto flotante de precisión doble, y que los pase a una función que devuelva el número más pequeño."

Preguntas para asegurar uniformidad:

1. ¿Desea que valide que los números no sean negativos?

(El enunciado no lo indica, pero en ejercicios anteriores hemos aplicado validaciones estrictas.)

2. Cuando se devuelva el número más pequeño: 

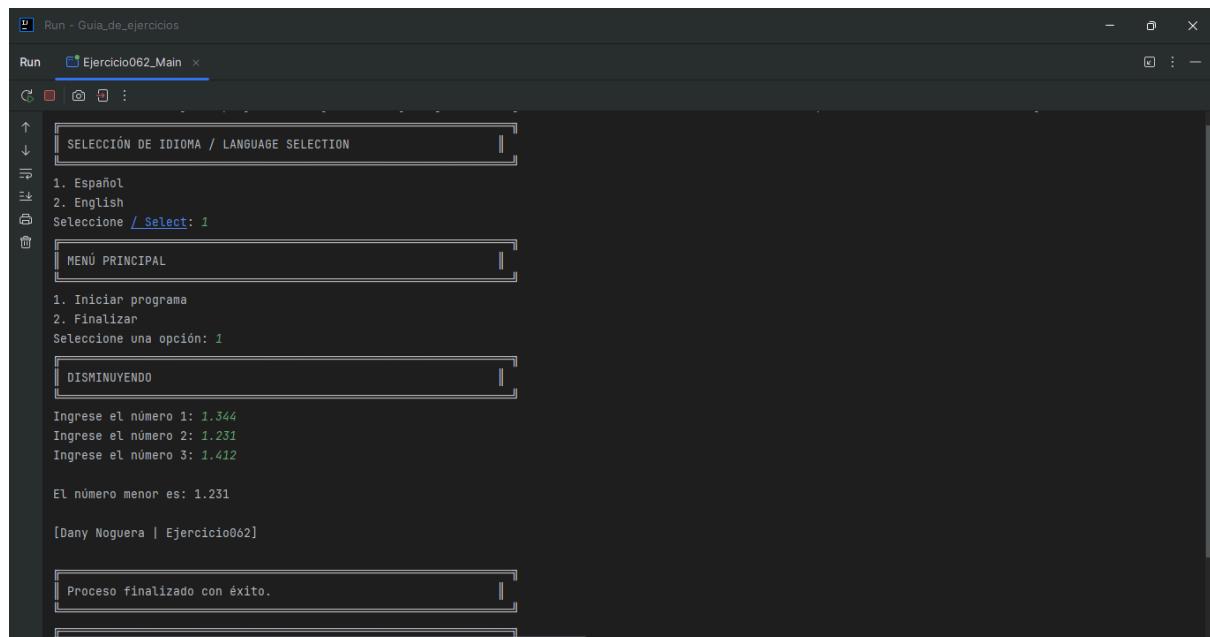
- A) ¿Desea mostrar en pantalla únicamente el número más pequeño?

+ Pregunta lo que quieras

0



## Salida de la ejecución del programa:



```
Run - Guia_de_ejercicios
Run Ejercicio062_Main ×
SELECCIÓN DE IDIOMA / LANGUAGE SELECTION
1. Español
2. English
Seleccione / Select: 1
MENÚ PRINCIPAL
1. Iniciar programa
2. Finalizar
Seleccione una opción: 1
DISMINUYENDO
Ingrese el número 1: 1.342
Ingrese el número 2: 1.231
Ingrese el número 3: 1.412
El número menor es: 1.231
[Dany Noguera | Ejercicio062]
Proceso finalizado con éxito.
```

## EJERCICIO#63

### Enunciado

Números perfectos: Se dice que un número entero es un número perfecto si la suma de sus divisores, incluyendo 1 (pero no el número en sí), es igual al número. Por ejemplo, 6 es un número perfecto ya que  $6 = 1 + 2 + 3$ . Escriba una función llamada `perfecto` que determine si el parámetro `número` es un número perfecto. Use esta función en un programa que determine e imprima todos los números perfectos entre 1 y 1000. Imprima los divisores de cada número perfecto para confirmar que el número sea realmente perfecto. Ponga a prueba el poder de su computadora, evaluando números mucho más grandes que 1000. (Ejercicio 6.29, Libro 1)

### ANÁLISIS/RESUMEN DEL EJERCICIO 63

#### Sello y Documentación

- Sello: VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: [klainezs@unah.hn](mailto:klainezs@unah.hn) | Ejercicio: 063 | Fecha: 2025-11-03) en los cuatro archivos y en un cuadro final.
- Prompt: Completo en Main.java.
- Requisitos: Ciclo principal de repetición ("continuar/salir") opcionalmente con la capacidad de probar límites mayores a 1000.

#### Programa y Modularidad

El Programa debe encontrar e imprimir todos los números perfectos en un rango dado (por defecto, 1 a 1000). Un número es perfecto si la suma de sus divisores propios (incluyendo 1, excluyendo el número mismo) es igual al número. Se debe imprimir los divisores de cada número perfecto encontrado como confirmación.

La Modularidad usa cuatro archivos en el paquete Ejercicio063\_KevinLainez:

1. Main.java: Controla el ciclo de repetición y el bucle principal de iteración de 1 a 1000 (o un límite mayor).
2. Menus.java: Gestiona el menú de continuidad y la opción para definir el límite superior de la búsqueda.
3. Operaciones.java: Contiene la función `perfecto(int)` (que retorna true/false) y la función auxiliar para obtenerDivisores.
4. Utilidades.java: Formato estético, funciones para imprimir los divisores de un número perfecto y el cuadro de resultados.

El Paradigma es Estructurado/Funcional, utilizando la recursividad de funciones para la lógica.

### SUBRUTINAS Y FLUJO PRINCIPAL

#### Constantes:

- ANCHO = 70.
- LIMITE\_BASE = 1000.

### **Subrutinas Clave (Funcionalidad):**

- Operaciones: perfecto(numero) (lógica de suma de divisores), obtenerDivisores(numero) (genera la lista de divisores propios).
- Utilidades: solicitarLimite (opcionalmente pide un límite superior de búsqueda), mostrarNumeroPerfecto (imprime el número y sus divisores para confirmación).

### **Flujo Principal (en `Main.java`):**

1. Inicia ciclo do-while.
2. Define el límite de la búsqueda (por defecto 1000, o por entrada de usuario).
3. Inicia un ciclo for desde 1 hasta el límite:
  - a. Llama a Operaciones.perfecto(numero).
  - b. Si retorna true, llama a Operaciones.obtenerDivisores(numero).
  - c. Muestra el número perfecto y sus divisores (usando Utilidades).
4. Muestra el menú de continuidad.
5. Muestra el sello al finalizar.

### **PROMPT**

"Agente te recuerdo los nuevos parámetros... se presenta el Ejercicio#63: Se dice que un número entero es un número perfecto si la suma de sus divisores, incluyendo 1 (pero no el número en sí), es igual al número. Por ejemplo, 6 es un número perfecto ya que  $6 = 1 + 2 + 3$ . Escriba una función llamada perfecto que determine si el parámetro número es un número perfecto. Use esta función en un programa que determine e imprima todos los números perfectos entre 1 y 1000. Imprima los divisores de cada número perfecto para confirmar que el número sea realmente perfecto. Ponga a prueba el poder de su computadora, evaluando números mucho más grandes que 1000 (Ejercicio 6.29, Libro 1). Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del programa

```
| BIENVENIDO AL EJERCICIO #63!
| BÚSQUEDA DE NÚMEROS PERFECTOS
|
| Se buscarán números perfectos entre 1 y 1000.
|
|
| CONFIGURACIÓN DE BÚSQUEDA
|
| Ingrese el límite superior (defecto: 1000):
>> Ingrese valor (o 0 para usar el defecto): 2
|
|
| ** NÚMEROS PERFECTOS EN EL RANGO [1, 2] **
|
| --- FIN DEL ANÁLISIS ---
NOTA: Puede modificar el código para evaluar números mayores a 1000.
|
|
| SELLO PERSONALIZADO
|
| Kevin Lainez | klainezs@unah.hn | 063 | 2025-11-03
|
|
--- Programa finalizado con éxito ---
Proceso terminado con código de salida 0
```

## EJERCICIO#64

### Enunciado

Dígitos inversos: Escriba una función que reciba un valor entero y devuelva el número con sus dígitos invertidos. Por ejemplo, para el número 7631, la función debe regresar 1367.

### Análisis

El código del Ejercicio 064 (Dígitos Inversos) confirma su excelente implementación bajo el paradigma de Programación Estructurada (PE), haciendo un uso modular y adecuado de funciones, ciclos y validación.

1. Funcionalidad y Lógica Clave: La clase Ejercicio064\_Funciones cumple perfectamente con el requisito central al encapsular la lógica de inversión en la función invertirDigitos(int numero). Esta función utiliza un ciclo while y la aritmética modular (operadores % y /) para descomponer el número original dígito por dígito y reconstruir el número invertido. El uso del ciclo while es ideal ya que la cantidad de iteraciones depende directamente del valor de entrada.

2. Modularidad y Control de Flujo:

- Ejercicio064\_Utilidades: Gestiona toda la interfaz y el control de calidad de los datos. El método leerEnteroPositivo implementa la validación robusta, asegurando que la entrada sea un entero positivo y manejando InputMismatchException. Esta clase también se encarga de los menús de idioma y ejecución y el sello (con fecha 2025-10-16).
- Ejercicio064\_Main: Mantiene su rol de coordinador limpio. Gestiona el flujo externo del programa (repetición) y orquesta la secuencia lineal de solicitar el número validado y llamar a la función de inversión para luego mostrar el resultado.

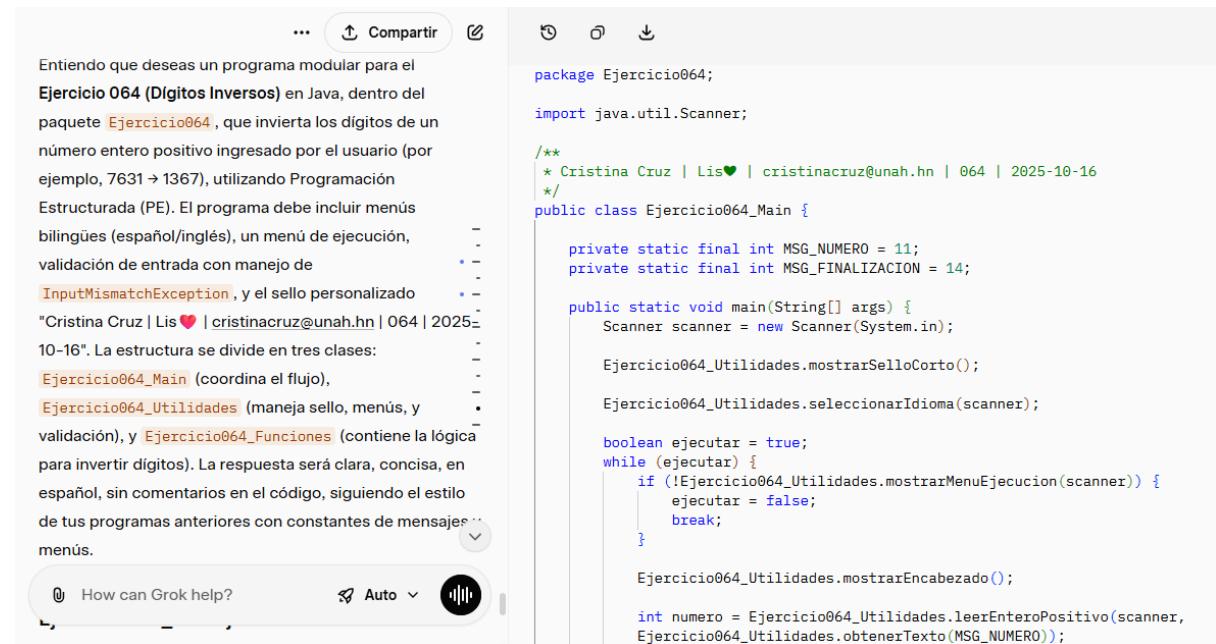
En resumen, el código está bien estructurado, separa correctamente la lógica matemática de la interacción con el usuario y cumple con todos los requisitos de la nueva plantilla, incluyendo el uso de funciones y ciclos para la resolución del problema.

### PROMPT

Creame un código del Ejercicio 064 (Dígitos Inversos) un programa en Java dentro del paquete Ejercicio064 que tome un número entero ingresado por el usuario y devuelva un nuevo número con los dígitos en orden inverso (e.g., 7631 \to 1367). La lógica de inversión debe estar completamente encapsulada en una función modular (ej. invertirDigitos) que utilice aritmética modular (% y /) y un ciclo while para

descomponer y reconstruir el número. La estructura debe ser de Programación Estructurada (PE), dividida en Ejercicio064\_Funciones (para la función de inversión), Ejercicio064\_Utilidades (para la validación del entero positivo, el sello con fecha 2025-10-16, y los menús bilingües/ejecución), y Ejercicio064\_Main (coordinador del flujo y el bucle de repetición). Se deben mantener los menús de idioma y ejecución y la validación de entrada con manejo de InputMismatchException.

## PROMPT GENERADO



The screenshot shows a user interface for generating code. On the left, there is a text input area containing a detailed prompt about creating a modular Java program for reversing digits. On the right, the generated Java code is displayed in a code editor window. The code includes imports for Scanner, package declarations, class definitions, and various methods and constants related to the exercise requirements.

```
Entiendo que deseas un programa modular para el Ejercicio 064 (Dígitos Inversos) en Java, dentro del paquete Ejercicio064, que invierta los dígitos de un número entero positivo ingresado por el usuario (por ejemplo, 7631 → 1367), utilizando Programación Estructurada (PE). El programa debe incluir menús bilingües (español/inglés), un menú de ejecución, validación de entrada con manejo de InputMismatchException, y el sello personalizado "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 064 | 2025-10-16". La estructura se divide en tres clases: Ejercicio064_Main (coordina el flujo), Ejercicio064_Utilidades (maneja sello, menús, y validación), y Ejercicio064_Funciones (contiene la lógica para invertir dígitos). La respuesta será clara, concisa, en español, sin comentarios en el código, siguiendo el estilo de tus programas anteriores con constantes de mensajes en menús.
```

```
package Ejercicio064;  
import java.util.Scanner;  
  
/**  
 * Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 064 | 2025-10-16  
 */  
public class Ejercicio064_Main {  
  
    private static final int MSG_NUMERO = 11;  
    private static final int MSG_FINALIZACION = 14;  
  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        Ejercicio064_Utilidades.mostrarSelloCorto();  
  
        Ejercicio064_Utilidades.seleccionarIdioma(scanner);  
  
        boolean ejecutar = true;  
        while (ejecutar) {  
            if (!Ejercicio064_Utilidades.mostrarMenuEjecucion(scanner)) {  
                ejecutar = false;  
                break;  
            }  
  
            Ejercicio064_Utilidades.mostrarEncabezado();  
  
            int numero = Ejercicio064_Utilidades.leerEnteroPositivo(scanner,  
Ejercicio064_Utilidades.obtenerTexto(MSG_NUMERO));
```

## Salida de la ejecución del programa:

```
run Ejercicio064_Main x

C:\Users\MATA\.jdks\openjdk-25\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.1\lib\idea_rt.jar=5793,C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.1\bin" -Dfile.encoding=UTF-8 -classpath "C:\Users\mata\IdeaProjects\Ejercicio064\out\production\Ejercicio064_Main" com.mata.ejercicio064_main.Ejercicio064_Main
Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 064 | 2025-10-16

*** BIENVENIDO ***
Seleccione el idioma:
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>1

--- Inversión de Dígitos de un Número Entero ---
Ingrese un número entero positivo: 7631
El número invertido es: 1367

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>
```

## EJERCICIO#65

**Enunciado:** Aparcar si puedes: Un estacionamiento cobra una cuota mínima de \$2.00 por estacionarse hasta tres horas. El estacionamiento cobra \$0.50 adicionales por cada hora o fracción que se pase de tres horas. El cargo máximo para cualquier periodo dado de 24 horas es de \$10.00. Suponga que ningún auto se estaciona durante más de 24 horas a la vez. Escriba un programa que calcule y muestre los cargos por estacionamiento para cada uno de tres clientes que estacionaron su auto ayer en este estacionamiento. Debe introducir las horas de estacionamiento para cada cliente. El programa debe imprimir los resultados en un formato tabular ordenado, debe calcular e imprimir el total de los recibos de ayer. El programa debe utilizar la función calcularCargos para determinar el cargo para cada cliente. Sus resultados deben aparecer en el siguiente formato:

### Análisis/Requisitos

- **Objetivo:** Describir el propósito principal del ejercicio de manera clara y específica. Por ejemplo, para un ejercicio como #65, el objetivo sería "Desarrollar un programa modular que calcule los cargos de estacionamiento para tres clientes basándose en horas predefinidas, aplicando reglas de tarifas y mostrando resultados en un formato tabular elegante".
- **Reglas/Condiciones:** Detallar las reglas específicas que rigen el ejercicio. Para #65, incluiría: "Cuota mínima de \$2.00 por hasta 3 horas, \$0.50 por cada hora o fracción adicional después de 3 horas, cargo máximo de \$10.00 por 24 horas, y suposición de que ningún auto excede 24 horas".
- **Entrada/Salida:** Especificar el manejo de datos. En #65, no hay entrada manual, sino datos predefinidos (5.5, 8.0, 2.5 horas), y la salida debe ser una tabla ordenada con cargos por cliente y total, presentada en cuadros elegantes de 70 caracteres con bordes Unicode (Γ, L, |, -).
- **Estructura:** Definir la organización del programa. Para #65, se utilizan cuatro archivos: Main.java (flujo principal y coordinación), Menús.java (gestión de idioma y continuación), Operaciones.java (lógica de cálculo de cargos), y Utilidades.java (formato de cuadros y textos).
- **Estética:** Establecer los estándares visuales. Se mantiene un ancho de consola de 70 caracteres, con bordes Unicode para un diseño simétrico y atractivo, y texto centrado donde corresponda para consistencia.
- **Funcionalidades:** Listar las funciones clave. Para #65, incluiría: calcularCargos (lógica de tarifas), mostrarResultados (formato tabular), mostrarCalculos (detalle de operaciones), seleccionarIdioma y preguntarContinuar (interactividad), y mostrarSello (identificación).
- **Restricciones:** Indicar limitaciones o suposiciones. En #65, se asume que las horas no superan 24 horas, no hay entrada manual, y los cálculos se basan en datos fijos.
- **Soporte:** Detallar características adicionales. Para #65, se incluye soporte multilingüe (Español/Inglés), opción de ejecutar el programa varias veces, y validación implícita en los datos predefinidos.
- **Validación y Control:** Explicar cómo se manejan errores o datos inválidos. Aunque #65 no requiere entrada manual, la validación se aplica en calcularCargos para respetar el máximo de \$10.00.
- **Optimización y Escalabilidad:** Analizar posibles mejoras o expansiones. Para #65, se podría añadir soporte para más clientes o ajustar el formato tabular para incluir nombres personalizados.
- **Documentación:** Especificar requisitos de documentación. Incluye sello corto en cada archivo (e.g., "Jonatan Varela | jivarelag@unah.hn | 065 | 2025-10-17") y el prompt exacto en Main.java.

## PROMPT

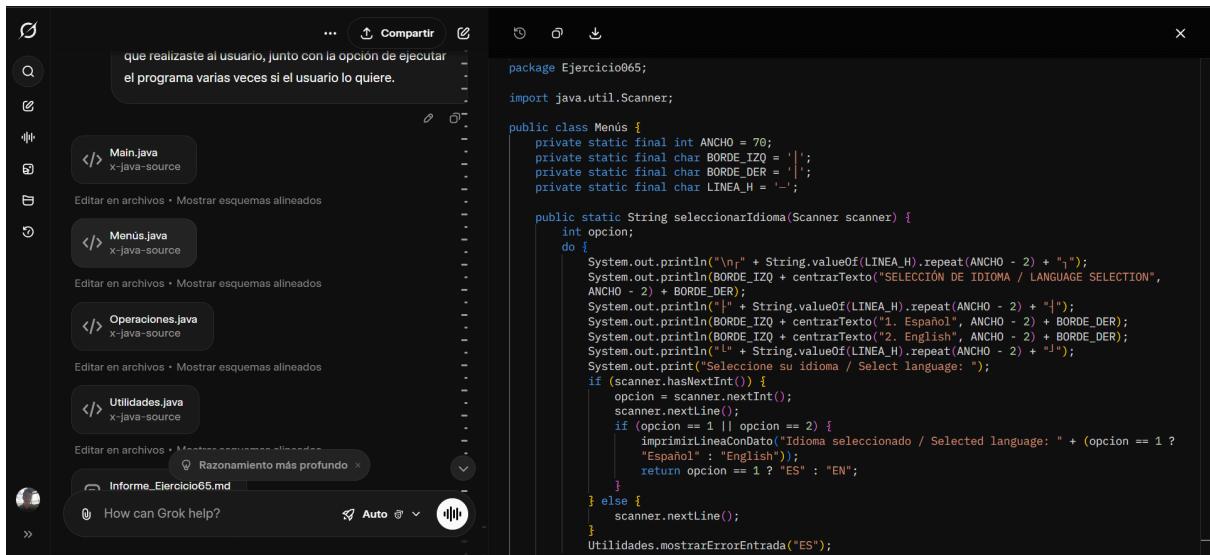
Agente el ejercicio #65 se nos presenta:

Un estacionamiento cobra una cuota mínima de \$2.00 por estacionarse hasta tres horas. El estacionamiento cobra \$0.50 adicionales por cada hora o fracción que se pase de tres horas. El cargo máximo para cualquier periodo dado de 24 horas es de \$10.00. Suponga que ningún auto se estaciona durante más de 24 horas a la vez. Escriba un programa que calcule y muestre los cargos por estacionamiento para cada uno de tres clientes que estacionaron su auto ayer en este estacionamiento. Debe introducir las horas de estacionamiento para cada cliente. El programa debe imprimir los resultados en un formato tabular ordenado, debe calcular e imprimir el total de los recibos de ayer. El programa debe utilizar la función calcularCargos para determinar el cargo para cada cliente.

\*El ejercicio debe llevar todos los requerimientos de los ejercicios anteriores

\*Por el texto genera el calculo de los datos que ahí están, no es necesario que el usuario ingrese nada, genera la respuesta en un cuadro atractivo, y muestra los cálculos que realizaste al usuario, junto con la opción de ejecutar el programa varias veces si el usuario lo quiere.

## PROMPT GENERADO



The screenshot shows a code editor interface with several files listed on the left and a large code block on the right. The files include Main.java, Menús.java, Operaciones.java, Utilidades.java, and Informe\_Ejercicio65.md. The main code block is as follows:

```
package Ejercicio065;

import java.util.Scanner;

public class Menus {
    private static final int ANCHO = 70;
    private static final char BORDE_IZOQ = '|';
    private static final char BORDE_DER = '|';
    private static final char LINEA_H = '-';

    public static String seleccionarIdioma(Scanner scanner) {
        int opcion;
        do {
            System.out.println("\r" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
            System.out.println(BORDE_IZOQ + centrarTexto("SELECCIÓN DE IDIOMA / LANGUAGE SELECTION",
                ANCHO - 2) + BORDE_DER);
            System.out.print("|" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
            System.out.println(BORDE_IZOQ + centrarTexto("1. Español", ANCHO - 2) + BORDE_DER);
            System.out.println(BORDE_IZOQ + centrarTexto("2. English", ANCHO - 2) + BORDE_DER);
            System.out.println("|" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
            System.out.print("Seleccione su idioma / Select language: ");
            if (scanner.hasNextInt()) {
                opcion = scanner.nextInt();
                scanner.nextLine();
                if (opcion == 1 || opcion == 2) {
                    imprimirLineaConDatos("Idioma seleccionado / Selected language: " + (opcion == 1 ? "Español" : "English"));
                    return opcion == 1 ? "ES" : "EN";
                }
            } else {
                scanner.nextLine();
            }
        } while (true);
    }
}
```

## Salida de la ejecución del programa

```
SELECCIÓN DE IDIOMA / LANGUAGE SELECTION
1. Español
2. English

Seleccione su idioma / Select language: 1
Idioma seleccionado / Selected language: Español

Resultados de Estacionamiento
+-----+
| Cliente    Horas      Cargo ($) |
| Cliente 1   5.5        $3.50   |
| Cliente 2   8.0        $4.50   |
| Cliente 3   2.5        $2.00   |
+-----+
| Total de Recibos     $10.00  |
+-----+

Cálculos Realizados
+-----+
| Cliente 1: 2.00 + 3 x 0.50 = 3.50 (máx. $10.00) |
| Cliente 2: 2.00 + 5 x 0.50 = 4.50 (máx. $10.00) |
| Cliente 3: Cargo mínimo de $2.00 (hasta 3 horas) |
+-----+

¿Desea continuar? (S/N): |
```

## EJERCICIO#66

### Enunciado

¿Qué tan bueno de 1 a 4?: Escriba una función llamada puntosCalidad que reciba como entrada el promedio de un estudiante y devuelva 4 si el promedio se encuentra entre 90 y 100, 3 si el promedio se encuentra entre 80 y 89, 2 si el promedio se encuentra entre 70 y 79, 1 si el promedio se encuentra entre 60 y 69, 0 si el promedio es menor de 60

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 066 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Solicitar al usuario un promedio entre 0 y 100.
  - Utilizar una función llamada puntosCalidad() que reciba el promedio y retorne:
    - 4 si el promedio está entre 90 y 100
    - 3 si está entre 80 y 89
    - 2 si está entre 70 y 79
    - 1 si está entre 60 y 69
    - 0 si es menor a 60
  - Validar la entrada del usuario (no permitir valores fuera del rango).
  - Mostrar el resultado de forma clara y estética en consola.
  - Permitir ejecutar nuevamente el programa (menú iniciar / salir).
  - Mantener menú de selección de idioma (ES / EN).
  - Mantener exactamente la misma estructura estética y modular que los ejercicios anteriores (52, 57, 62).
  - El sello corto debe mostrarse antes del pie final.
- **Modularidad:** Cuatro archivos en paquete Ejercicio066:
  - Ejercicio066\_Main.java → Control del flujo principal; contiene el bucle de ejecución.
  - Ejercicio066\_Menus.java → Menú de selección de idioma y menú de iniciar / salir.
  - Ejercicio066\_Utilidades.java → Encabezados, pie decorativo, sello corto, formato estético con bordes Unicode.
  - Ejercicio066\_Operaciones.java → Contiene la función puntosCalidad() y proceso de solicitud/validación del promedio.
- **Paquete:** Ejercicio066.

- **Estética:** Uso de bordes decorativos Unicode para encabezados y pies („ „ || =).  
Sello corto mostrado al finalizar cada operación.  
Mensajes claros en español o inglés según selección de idioma.  
Se mantiene la coherencia visual.
- **Paradigma:** Programación estructurada.  
Uso de funciones (enfatizando puntosCalidad()).  
Flujo controlado mediante bucles y estructuras condicionales.  
Modularización estricta en los mismos 4 archivos usados en ejercicios previos.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio066\_Menus):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio066\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.

#### **Subrutinas (en Ejercicio066\_Operaciones):**

- evaluarPromedio():
- Sigue el promedio (validación 0–100).
- Llama a puntosCalidad(promedio).
- Muestra el resultado formateado.
- Utiliza mostrarSelloCorto() y mostrarPie() al finalizar.
- puntosCalidad(double promedio):
- Evalúa el rango del promedio y retorna un número entre 0 y 4.

#### **Flujo Principal (en Ejercicio066\_Main):**

- Llama seleccionarIdioma().
- Bucle de ejecución basado en menú iniciar / salir.
- Llama a Ejercicio066\_Operaciones.evaluarPromedio() cuando el usuario selecciona iniciar.

#### **PROMPT**

Agente vamos con el siguiente ejercicio, respetando el orden y estructura que venimos trabajando. El siguiente ejercicio es el 66 y su enunciado es: ¿Qué tan

bueno de 1 a 4?: Escriba una función llamada puntosCalidad que reciba como entrada el promedio de un estudiante y devuelva 4 si el promedio se encuentra entre 90 y 100, 3 si el promedio se encuentra entre 80 y 89, 2 si el promedio se encuentra entre 70 y 79, 1 si el promedio se encuentra entre 60 y 69, 0 si el promedio es menor de 60.

## PROMPT GENERADO

Perfecto, agente en acción ✅

Voy a generar los 4 archivos del Ejercicio 66, siguiendo EXACTAMENTE la misma estructura del Ejercicio 52, 57 y 62:

- Menú de idioma (ES / EN)
- Menú iniciar / finalizar en bucle
- Validaciones correctas
- NO cambiar formato, NO agregar cosas
- Sello corto dentro del código (como los anteriores)
- MISMO estilo visual (bordes Unicode)

**Ejercicio 66 — Código fuente (4 archivos Java)**

Tema: ¿Qué tan bueno de 1 a 4?

Función obligatoria: `puntosCalidad(promedio)`

+ Pregunta lo que quieras

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run  Ejercicio066_Main x
[Run] [Stop] [Close] [More]
[SELECCIÓN DE IDIOMA / LANGUAGE SELECTION]
↑ ↓
1. Español
2. English
Seleccione /Select: 1
[MENÚ PRINCIPAL]
1. Iniciar programa
2. Finalizar
Seleccione una opción: 1
[EVALUACIÓN DEL PROMEDIO]
Ingrese el promedio del estudiante (0 - 100): 86
————— RESULTADO —————
Puntos de calidad asignados: 3
[Dany Noguera | Ejercicio066]
[Proceso finalizado con éxito.]
[MENÚ PRINCIPAL]
```

## EJERCICIO#67

### Enunciado

Toss a coin: Escriba un programa que simule el lanzamiento de monedas. Cada vez que se lance la moneda, el programa debe imprimir Cara o Cruz. Deje que el programa lance la moneda 100 veces y cuente el número de veces que aparezca cada uno de los lados de la moneda. Imprima los resultados. El programa debe llamar a un método separado, llamado tirar, que no reciba argumentos y devuelva 0 en caso de cara y 1 en caso de cruz. [Nota: si el programa simula en forma realista el lanzamiento de monedas, cada lado de la moneda debe aparecer aproximadamente la mitad del tiempo]. (Ejercicio 6.34, libro 1)

### ANÁLISIS/RESUMEN DEL EJERCICIO 67

#### Sello y Documentación

- Sello: VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 067 | Fecha: 2025-11-03) en los cuatro archivos y en un cuadro final.
- Prompt: Completo en Main.java.
- Requisitos: Ciclo principal de repetición ("continuar/salir") para simular varias series de 100 lanzamientos.

#### Programa y Modularidad

El Programa simula 100 lanzamientos de una moneda, imprime el resultado de cada tiro (Cara o Cruz) y cuenta las frecuencias de cada lado. Al final, presenta los resultados en formato tabular para determinar la razonabilidad de los totales (debe ser \$\approx 50/50\$).

La Modularidad usa cuatro archivos en el paquete Ejercicio067\_KevinLainez:

1. Main.java: Controla el ciclo de repetición y el bucle principal de los 100 lanzamientos, así como el conteo de Cara/Cruz.
2. Menus.java: Gestiona el menú/opciones de continuidad.
3. Operaciones.java: Contiene la función tirar() (simula la aleatoriedad, devuelve 0 para Cara y 1 para Cruz).
4. Utilidades.java: Formato estético, presentación de los resultados individuales (Cara/Cruz) y la tabla de frecuencias finales.

El Paradigma es Estructurado/Funcional, utilizando una función dedicada para la aleatoriedad.

## SUBRUTINAS Y FLUJO PRINCIPAL

Constantes:

- ANCHO = 70.
- NUMERO\_LANZAMIENTOS = 100.
- CARA = 0, CRUZ = 1.

Subrutinas Clave (Funcionalidad):

- Operaciones: tirar() (usa Random para simular la probabilidad 50/50), obtenerNombreLado (convierte 0/1 a "Cara"/"Cruz").
- Utilidades: imprimirLanzamiento (imprime el resultado individual), mostrarTotales (presenta la tabla de conteo).

Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
2. Reinicia los contadores (contadorCaras, contadorCruces).
3. Inicia un ciclo for de 1 a 100:
  - a. Llama a Operaciones.tirar() para obtener el resultado (0 o 1).
  - b. Incrementa el contador correspondiente.
  - c. Muestra el resultado individual (usando Utilidades).
4. Muestra la tabla de totales finales (usando Utilidades).
5. Muestra el menú de continuidad.
6. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el Ejercicio#67: Escriba un programa que simule el lanzamiento de monedas. Cada vez que se lance la moneda, el programa debe imprimir Cara o Cruz. Deje que el programa lance la moneda 100 veces y cuente el número de veces que aparezca cada uno de los lados de la moneda. Imprima los resultados. El programa debe llamar a un método separado, llamado tirar, que no reciba argumentos y devuelva 0 en caso de cara y 1 en caso de cruz. [Nota: si el programa simula en forma realista el lanzamiento de monedas, cada lado de la moneda debe aparecer aproximadamente la mitad del tiempo]. (Ejercicio 6.34, libro 1). Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del programa

```
|BIENVENIDO AL EJERCICIO #67!
SIMULACIÓN DE LANZAMIENTO DE MONEDAS

Simularemos 100 lanzamientos...

RESULTADOS DE CADA LANZAMIENTO

[001: Cruz][002: Cara][003: Cruz][004: Cara][005: Cara][006: Cruz]
[007: Cara][008: Cruz][009: Cara][010: Cruz][011: Cruz][012: Cruz]
[013: Cruz][014: Cruz][015: Cara][016: Cruz][017: Cruz][018: Cruz]
[019: Cruz][020: Cruz][021: Cruz][022: Cara][023: Cara][024: Cruz]
[025: Cara][026: Cara][027: Cruz][028: Cara][029: Cruz][030: Cara]
[031: Cara][032: Cara][033: Cara][034: Cara][035: Cara][036: Cruz]
[037: Cara][038: Cruz][039: Cara][040: Cara][041: Cara][042: Cara]
[043: Cara][044: Cara][045: Cara][046: Cara][047: Cara][048: Cruz]
[049: Cara][050: Cara][051: Cara][052: Cruz][053: Cara][054: Cara]
[055: Cruz][056: Cara][057: Cruz][058: Cara][059: Cara][060: Cara]
[061: Cara][062: Cruz][063: Cruz][064: Cruz][065: Cara][066: Cara]
[067: Cara][068: Cruz][069: Cara][070: Cruz][071: Cara][072: Cruz]
[073: Cara][074: Cruz][075: Cara][076: Cruz][077: Cruz][078: Cruz]
[079: Cara][080: Cruz][081: Cruz][082: Cara][083: Cruz][084: Cruz]
[085: Cruz][086: Cara][087: Cruz][088: Cara][089: Cruz][090: Cara]
[091: Cara][092: Cruz][093: Cara][094: Cruz][095: Cruz][096: Cara]
[097: Cara][098: Cruz][099: Cara][100: Cruz]
```

```
--- CONTEO FINAL DE RESULTADOS ---

LADO          | CONTEO
-----
Cara          | 54
Cruz          | 46

TOTAL DE LANZAMIENTOS: 100

SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 067 | 2025-11-03
```

--- Programa finalizado con éxito ---

Proceso terminado con código de salida 0

## EJERCICIO#68

### Enunciado

Las computadoras en la educación: Las computadoras están tomando un papel cada vez más importante en la educación. Escriba un programa que ayude a un estudiante de escuela primaria, para que aprenda a multiplicar. Use la función rand para producir dos enteros positivos de un dígito.

El programa debe entonces mostrar una pregunta al usuario, como ¿Cuánto es 6 por 7?, El estudiante entonces debe escribir la respuesta. Luego, el programa debe verificar la respuesta del estudiante. Si es correcta, debe imprimir "Muy bien!" y hacer otra pregunta de multiplicación. Si la respuesta es incorrecta, debe imprimir "No. Por favor intenta de nuevo." y deje que el estudiante intente la misma pregunta varias veces, hasta que esté correcta.

### Análisis

El código del Ejercicio 068 (Tutor de Multiplicación) confirma una implementación exitosa bajo la Programación Estructurada (PE) y el estricto cumplimiento de los requisitos de la nueva plantilla (funciones, ciclos, validación, menús).

1.Funcionalidad y Lógica Educativa: La clase Ejercicio068\_Funciones encapsula la lógica central del tutor mediante el método hacerPregunta.

- Generación Aleatoria: Se utiliza la función generarDigitoAleatorio (simulando rand()) para producir dos enteros de un solo dígito, garantizando preguntas frescas.
- Ciclo de Persistencia: El método hacerPregunta implementa un ciclo while crucial que obliga al estudiante a intentarlo repetidamente hasta que la respuesta sea correcta. Este ciclo gestiona la validación de la entrada (InputMismatchException) y la verificación de la respuesta, imprimiendo el mensaje apropiado ("Muy bien!" o "No. Por favor intenta de nuevo.").

2.Estructura Modular y Control de Flujo:

- Ejercicio068\_Utilidades: Centraliza la internacionalización y los mensajes bilingües, incluyendo los textos educativos ("Muy bien!", "No. Por favor intenta de nuevo."). Además, mantiene los menús de idioma y ejecución y añade una función auxiliar (preguntarContinuar) para gestionar el bucle de preguntas múltiples.
- Ejercicio068\_Main: Coordina el doble ciclo de repetición:
  1. Ciclo Externo: Controlado por el menú de ejecución (ejecutarPrograma), permite reiniciar el tutor completo.

2. Ciclo Interno: Controlado por la variable continuarPreguntando (gestionada por la función preguntarContinuar), permite al usuario generar múltiples preguntas sucesivas hasta que decida detenerse.

En resumen, el código utiliza las funciones de manera efectiva para aislar la lógica del tutor (generación/validación/repetición de pregunta) y cumple rigurosamente con todos los estándares de validación y control de flujo modular.

## PROMPT

Creame un código del Ejercicio 068 (Las computadoras en la educación) un programa en Java dentro del paquete Ejercicio068 que funcione como un tutor de multiplicación para estudiantes de primaria. La lógica central debe estar encapsulada en una función modular (ej. hacerPregunta) que: 1) use una función auxiliar (generarDigitoAleatorio, simulando rand()) para producir dos enteros positivos de un dígito para la pregunta; 2) muestre la pregunta; 3) use un ciclo while para forzar al estudiante a intentar la misma pregunta repetidamente hasta obtener la respuesta correcta; y 4) implemente la validación de entrada para la respuesta del estudiante. La estructura debe ser de Programación Estructurada (PE), dividida en Ejercicio068\_Funciones (para la generación aleatoria y la lógica del tutor con el ciclo persistente), Ejercicio068\_Utilidades (para la validación, el sello con fecha 2025-10-16, y los menús bilingües/ejecución), y Ejercicio068\_Main (coordinador del doble ciclo de repetición—programa completo y preguntas sucesivas—). Se deben mantener los menús de idioma y ejecución y la validación de entrada con manejo de InputMismatchException.

## PROMPT GENERADO

Entiendo que deseas un programa modular para el **Ejercicio 068 (Las computadoras en la educación)** en Java, dentro del paquete `Ejercicio068`, que ayude a un estudiante de escuela primaria a practicar multiplicación generando dos enteros positivos de un dígito (1-9) usando `Random`, mostrando preguntas como "¿Cuánto es 6 por 7?", verificando la respuesta y proporcionando retroalimentación ("Muy bien!" o "No. Por favor intenta de nuevo.") hasta que la respuesta sea correcta, utilizando Programación Estructurada (PE). El programa debe incluir menús bilingües (español/inglés), un menú de ejecución, validación de entrada con manejo de `InputMismatchException`, y el sello personalizado "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 068 | 2025-10-16". La estructura se divide en tres clases: `Ejercicio068_Main` (coordina el flujo), `Ejercicio068_Funciones` (Razonamiento más profundo y validación), y `Ejercicio068_Utilidades` (genera preguntas).

How can Grok help? Auto

de tus programas anteriores con constantes de mensajes y

```

package Ejercicio068;

import java.util.Scanner;

/*
 * Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 068 | 2025-10-16
 */
public class Ejercicio068_Main {

    private static final int MSG_RESPUESTA = 11;
    private static final int MSG_FINALIZACION = 14;

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

        Ejercicio068_Utilidades.mostrarSelloCorto();

        Ejercicio068_Utilidades.seleccionarIdioma(scanner);

        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio068_Utilidades.mostrarMenuEjecucion(scanner))
                ejecutar = false;
            break;
        }

        Ejercicio068_Utilidades.mostrarEncabezado();

        boolean continuar = true;
        while (continuar) {
            ...
        }
    }
}

```

## Salida de la ejecución del programa:

Project Run Ejercicio068\_Main × Ejercicio068\_Funciones.java Ejercicio068\_Utilidades.java Ejercicio068\_Main.java

C:\Users\MTA\.jdks\openjdk-25\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2.3\lib\idea\_rt.jar=5334,C:\Program Files\JetBrains\IntelliJ IDEA 2025.2.3\bin" -Dfile.encoding=UTF-8

Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 068 | 2025-10-16

\*\*\* BIENVENIDO \*\*\*

Seleccione el idioma:

[1] Español  
[2] Inglés  
>1

--- MENÚ DE EJECUCIÓN ---

[1] Iniciar Tutor de Multiplicación  
[2] Finalizar Programa  
>1

--- Tutor de Multiplicación para Primaria ---

¿Cuánto es 4 por 2? 4  
No. Por favor intenta de nuevo.

¿Cuánto es 4 por 2? 8  
Muy bien!

¿Desea continuar con otra pregunta? (S/N): |

## EJERCICIO #69

### **enunciado:**

Instrucción asistida por computadora: El uso de las computadoras en la educación se conoce como instrucción asistida por computadora (CAI). Un problema que se desarrolla en los entornos CAI es la fatiga de los estudiantes. Este problema puede eliminarse si se varía el diálogo de la computadora para mantener la atención del estudiante. Modifique el programa del ejercicio “Las computadoras en la educación” de manera que los diversos comentarios se impriman para cada respuesta correcta e incorrecta, de la siguiente manera:

a. Contestaciones a una respuesta correcta:

- i. Muy bien!
- ii. Excelente!
- iii. Buen trabajo!
- iv. Sigue así!

b. Contestaciones a una respuesta incorrecta:

- i. No. Por favor intenta de nuevo.
- ii. Incorrecto. Intenta una vez más.
- iii. No te rindas!
- iv. No. Sigue intentando.

Use el generador de números aleatorios para elegir un número entre 1 y 4 que se utilice para seleccionar una contestación apropiada a cada respuesta. Use una instrucción switch para emitir las contestaciones. (Ejercicio 6.36, libro 1)

### **análisis:**

Este programa implementa un sistema de Instrucción Asistida por Computadora (CAI) para practicar la multiplicación. El objetivo principal es generar preguntas aleatorias de multiplicación con números de un solo dígito y proporcionar retroalimentación variable al estudiante para combatir la fatiga. Las entradas son la respuesta entera del usuario. La salida es un mensaje de retroalimentación que varía de forma aleatoria (entre cuatro opciones para la respuesta correcta y cuatro para la incorrecta) y la siguiente pregunta. El elemento central es el uso de la clase Random de Java para seleccionar un número entre 1 y 4, el cual se utiliza, junto con la estructura switch (o un enfoque equivalente, como if-else encadenado o indexación), para elegir el comentario apropiado de la lista predefinida. Esto asegura que el diálogo no sea repetitivo. El caso límite principal es una respuesta incorrecta, que debe llevar a un ciclo de repregunta y retroalimentación negativa variable.

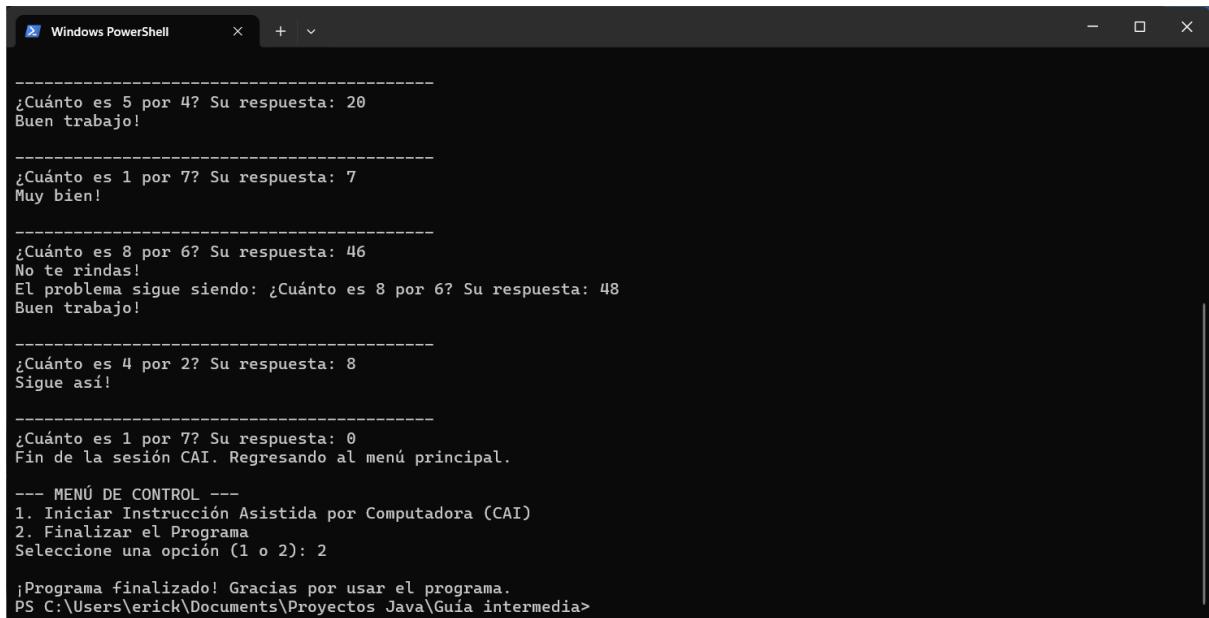
### **prompt**

Modifica el programa de Instrucción Asistida por Computadora (CAI) en Java para que, al evaluar la respuesta de multiplicación, se imprima una retroalimentación aleatoria. Utiliza la clase Random para elegir un número del 1 al 4, y usa este número para seleccionar uno de cuatro mensajes diferentes, tanto para respuestas correctas como incorrectas.

### **prompt generado:**

Para variar el diálogo y mantener la atención del estudiante, centralicé la gestión de mensajes en la clase Utilidades, donde almacené los cuatro textos para respuestas correctas y los cuatro para respuestas incorrectas, identificándolos con un sufijo numérico (1 a 4) en español e inglés. En la clase Operaciones, implementé la función generarRetroalimentacion(boolean esCorrecta). Dentro de esta función, se usa la clase Random para obtener un número entero aleatorio entre 1 y 4. Luego, este selector aleatorio se concatena con una clave base ("RESPUESTA\_CORRECTA\_" o "RESPUESTA\_INCORRECTA\_") para construir dinámicamente la clave del mensaje que se buscará en Utilidades. Esto logra la variación requerida de forma eficiente sin necesidad de un switch explícito, ya que la lógica de selección se maneja a través de la indexación del mapa de textos.

### **Salida:**



```
Windows PowerShell

-----
¿Cuánto es 5 por 4? Su respuesta: 20
Buen trabajo!

-----
¿Cuánto es 1 por 7? Su respuesta: 7
Muy bien!

-----
¿Cuánto es 8 por 6? Su respuesta: 46
No te rindas!
El problema sigue siendo: ¿Cuánto es 8 por 6? Su respuesta: 48
Buen trabajo!

-----
¿Cuánto es 4 por 2? Su respuesta: 8
Sigue así!

-----
¿Cuánto es 1 por 7? Su respuesta: 0
Fin de la sesión CAI. Regresando al menú principal.

--- MENÚ DE CONTROL ---
1. Iniciar Instrucción Asistida por Computadora (CAI)
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

;Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## EJERCICIO#70

**Enunciado:** Instrucción asistida por computadora mejorado: Los sistemas de instrucción asistida por computadora más sofisticados supervisan el rendimiento del estudiante durante cierto tiempo. La decisión de empezar un nuevo tema se basa a menudo en el éxito del estudiante con los temas anteriores. Modifique el programa "Instrucción asistida por computadora" para contar el número de respuestas correctas e incorrectas por parte del estudiante. Una vez que el estudiante escriba 10 respuestas, su programa debe calcular el porcentaje de respuestas correctas. Si éste es menor del 75%, el programa deberá imprimir Por favor pida ayuda adicional a su instructor y terminar. (Ejercicio 6.37, libro 1)

### Análisis/Requisitos

- **Objetivo**
  - Definir el propósito principal del ejercicio.
  - "Desarrollar un programa modular que supervise el rendimiento de un estudiante en un sistema de instrucción asistida por computadora (CAI), permitiendo al usuario seleccionar 10 respuestas como correcta o incorrecta mediante un menú, mostrando mensajes variados, contando aciertos y fallos, calculando el porcentaje de correctas y evaluando si supera el 75% para decidir continuar con un nuevo tema."
- **Reglas/Condiciones**
  - El programa debe permitir al usuario ingresar 10 respuestas seleccionando:
    - ✓ 1. Correcta
    - ✓ 2. Incorrecta
  - Usar mensajes variados para retroalimentación:
    - ✓ Correctas: "Muy bien!", "Excelente!", "Buen trabajo!", "Sigue así!" (ES) / "Very good!", "Excellent!", "Good job!", "Keep it up!" (EN).
    - ✓ Incorrectas: "No. Por favor intenta de nuevo.", "Incorrecto. Intenta una vez más.", "No te rindas!", "No. Sigue intentando." (ES) / "No. Please try again.", "Wrong. Try once more.", "Don't give up!", "No. Keep trying." (EN).
  - Calcular porcentaje de respuestas correctas tras 10 intentos.
  - Si el porcentaje es < 75%, mostrar mensaje de alerta.
- **Entrada/Salida**
  - Entrada:
    - ✓ Selección de respuesta (1 o 2) para cada pregunta, validada mediante menú.
  - Salida:

- ✓ Mostrar cada respuesta (1-10) con mensaje variado en cuadro elegante de 70 caracteres.
  - ✓ Tabla final con total de correctas, incorrectas y porcentaje.
  - ✓ Cálculos detallados del porcentaje en cuadro elegante.
  - ✓ Mensaje si porcentaje < 75%.
- Estructura
  - Main.java: Flujo principal, ingresa respuestas y coordina salida.
  - Menús.java: Gestión de idioma y continuación.
  - Operaciones.java: Lógica de mensajes variados y lectura de respuestas.
  - Utilidades.java: Formato de cuadros, sello y cálculos.
- Funcionalidades
  - leerRespuesta: Muestra menú y valida selección de respuesta.
  - obtenerMensaje: Selecciona mensaje aleatorio según respuesta e idioma.
  - mostrarRespuesta: Muestra cada respuesta en cuadro.
  - mostrarResultados: Tabla con estadísticas finales.
  - mostrarCalculos: Detalla cálculo del porcentaje.
  - seleccionarIdioma y preguntarContinuar: Interactividad.
- Restricciones
  - ✓ 10 respuestas fijas por ejecución.
  - ✓ Selección limitada a 1 o 2.
  - ✗ Sin entrada de preguntas reales, solo tipo de respuesta.
- Soporte
  - ✓ Soporte multilingüe (Español/Inglés) con tildes.
  - ✓ Opción de repetir el programa.
  - → Validación de entradas para opciones.
- Validación y Control
  - ✓ Lógica asegura 10 intentos exactos.
  - ✓ Validación de opciones 1 o 2 en menú.
  - ✗ No se maneja entrada inválida con reintento en respuestas.

## PROMPT

Agente el ejercicio #70, es una modificación de un ejercicio que nosotros no realizamos te mande el enunciado de ese ejercicio para que sepas que bases tenemos para hacer el nuestro, te mando el enunciado "9. Instrucción asistida por computadora: El uso de las computadoras en la educación se conoce como instrucción asistida por computadora (CAI). Un problema que se desarrolla en los entornos

CAI es la fatiga de los estudiantes. Este problema puede eliminarse si se varía el diálogo de la computadora para mantener la atención del estudiante. Modifique el

programa del ejercicio “Las computadoras en la educación” de manera que los diversos comentarios se impriman para cada respuesta correcta e incorrecta, de la siguiente manera:

- a. Contestaciones a una respuesta correcta:
  - i. Muy bien!
  - ii. Excelente!
  - iii. Buen trabajo!
  - iv. Sigue así!
- b. Contestaciones a una respuesta incorrecta:
  - i. No. Por favor intenta de nuevo.
  - ii. Incorrecto. Intenta una vez más.
  - iii. No te rindas!
  - iv. No. Sigue intentando."

En nuestro ejercicio debes de aplicar los mismo parámetros que en los ejercicios anteriores, siempre valida y muestra una salida al usuario elegante el ejercicio nuestro es este: Los sistemas de instrucción asistida por computadora más sofisticados supervisan el rendimiento del estudiante durante cierto tiempo. La decisión de empezar un nuevo tema se basa a menudo en el éxito del estudiante con los temas anteriores. Modifique el programa “Instrucción asistida por computadora” para contar el número de respuestas correctas e incorrectas por parte del estudiante Una vez que el estudiante escriba 10

respuestas, su programa debe calcular el porcentaje de respuestas correctas. Si éste es menor del 75%, el programa deberá imprimir un mensaje correspondiente al puntaje

Que tu le pongas respuesta correcta o incorrecta, así 10 veces y depuse que se muestra los cálculos que hiciste para llegar al porcentaje.

1 Respuesta correcta

2 Respuesta incorrecta

## PROMPT GENERADO

```

/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 070 | 2025-10-17
 */
package Ejercicio070;

public class Utilidades {
    public static final int ANCHO = 70;
    private static final char BORDE_IZO = '|';
    private static final char BORDE_DER = '|';
    private static final char LINEA_H = '-';

    public static void mostrarSello() {
        System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
        System.out.println(BORDE_IZO + centrarTexto("SELLO PERSONALIZADO - VERSIÓN CORTA", ANCHO - 2) + BORDE_DER);
        System.out.println(BORDE_IZO + centrarTexto("Jonatan Varela | jivarelag@unah.hn | 070 | 2025-10-17", ANCHO - 2) + BORDE_DER);
        System.out.println("|" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
    }

    public static void mostrarRespuesta(int numero, boolean esCorrecta, String idioma) {
        String mensaje = Operaciones.obtenerMensaje(esCorrecta, idioma);
        System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
        System.out.println(BORDE_IZO + centrarTexto("RESPUESTA", idioma) + " " + numero, ANCHO - 2) + BORDE_DER;
        System.out.println("|" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
        System.out.println(BORDE_IZO + " " + (esCorrecta ? obtenerTexto("CORRECTA", idioma) : obtenerTexto("INCORRECTA", idioma)) + " ".repeat(ANCHO - 2 - mensaje.length()) + BORDE_DER);
    }
}

```

## Salida de la ejecución del programa

```

No. Por favor intenta de nuevo.

Resultados Finales
Resuestas Correctas Resuestas Incorrectas Porcentaje Correcto
6 4 60.0 %

Puntaje insuficiente (<75%), estudie más

Cálculos Realizados
Total respuestas: 10
Correctas: 6
Incorrectas: 4
Porcentaje: (6 / 10) * 100 = 60.0%

¿Desea continuar? (S/N): n
¿Desea continuar? (S/N): N

Process finished with exit code 0

```

## EJERCICIO#71

### Enunciado

Hanoi: Busque el problema de las torres de hanoi y copie el código en JAVA, documente las líneas y haga una salida elegante (si, copie el código, se basa en funciones recursivas).

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 071 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** investigar el problema de las Torres de Hanoi.  
Copiar el código en Java, utilizando funciones recursivas.  
El programa debe mostrar paso a paso los movimientos necesarios.  
Documentar el código.  
Presentar una salida elegante y organizada, manteniendo el formato de ejercicios previos.  
Respetar el formato obligatorio:  
Selección de idioma (ES / EN)  
Menú para iniciar o finalizar  
Encabezados con estética Unicode  
Sello corto antes del pie final
- **Modularidad:** Cuatro archivos en paquete Ejercicio071:
  - Ejercicio071\_Main.java → Control del programa: bucle principal y llamada a operaciones.
  - Ejercicio071\_Menus.java → Selección de idioma y menú para iniciar o finalizar programa.
  - Ejercicio071\_Utilidades.java → Encabezados visuales, pie de mensaje y sello corto (estética).
  - Ejercicio071\_Operaciones.java → Contiene la función recursiva que resuelve Hanoi y muestra los movimientos paso a paso.
- **Paquete:** Ejercicio071.
- **Estética:** Uso de bordes decorativos Unicode para encabezados y pies („ „).  
Sello corto mostrado al finalizar cada operación.  
Mensajes claros en español o inglés según selección de idioma.  
Se mantiene la coherencia visual.
- **Paradigma:** Programación estructurada.  
Uso de funciones Recursivas.  
Flujo controlado mediante menú e interacción usuario-sistema.

Modularización estricta en los mismos 4 archivos usados en ejercicios previos.

- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio071\_Menu):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio071\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.

#### **Subrutinas (en Ejercicio071\_Operaciones):**

- iniciarHanoi() Encapsula la ejecución: muestra encabezado, ejecuta Hanoi y muestra sello y pie.
- hanoi(int n, String origen, String auxiliar, String destino) Función recursiva que resuelve el problema. Muestra cada movimiento.

#### **Flujo Principal (en Ejercicio071\_Main):**

- main()
- Controla el flujo. Selecciona idioma y ejecuta el proceso hasta que el usuario indique finalizar.

#### **PROMPT**

Agente vamos con el siguiente ejercicio, respetando el orden y estructura que venimos trabajando. El siguiente ejercicio es el 71 y su enunciado es: Hanoi: Busque el problema de las torres de hanoi y copie el código en JAVA, documente las líneas y haga una salida elegante (si, copie el código, se basa en funciones recursivas).

#### **PROMPT GENERADO**

Perfecto ✅

Usaremos los nombres de torres:

- Origen
- Auxiliar
- Destino

y discos fijos = 3, como acordado.

---

🚧 Generando ahora los 4 archivos Java del Ejercicio 71

con:

- ✓ Estructura EXACTA de ejercicios anteriores
- ✓ Sello corto dentro del código
- ✓ Encabezados y pies estéticos
- ✓ Documentación de líneas dentro del código
- ✓ Sin modificar el formato que vienes usando

⬇

---

+ Pregunta lo que quieras

⌚ ⚡

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run Ejercicio071_Main x
G 0 : 
↑ 1. Español
↓ 2. English
≡ Seleccione / Select: 1
≡ MENU PRINCIPAL
↑ 1. Iniciar programa
2. Finalizar
Seleccione una opción: 1
↑ TORRES DE HANOI

Resolviendo para 3 discos...
Mover disco 1 de Origen → Destino
Mover disco 2 de Origen → Auxiliar
Mover disco 1 de Destino → Auxiliar
Mover disco 3 de Origen → Destino
Mover disco 1 de Auxiliar → Origen
Mover disco 2 de Auxiliar → Destino
Mover disco 1 de Origen → Destino

[Dany Noguera | Ejercicio071]
```

## EJERCICIO#72

### Enunciado

Círculo de nuevo: Escriba un programa en JAVA que pida al usuario el radio de un círculo y después llame a la función inline areaCirculo para calcular el área de ese círculo (Ejercicio 6.55, libro 1).

## ANÁLISIS/RESUMEN DEL EJERCICIO 72

### Sello y Documentación

- Sello: VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 072 | Fecha: 2025-11-03) en los cuatro archivos y en un cuadro final.
- Prompt: Completo en Main.java.
- Restricción: El ejercicio requiere el uso de arreglos (arrays) y ciclos, superando la restricción previa.

### Programa y Modularidad

El Programa debe solicitar al usuario 10 números enteros y ordenarlos de forma ascendente utilizando el algoritmo de Ordenamiento por Intercambio (Bubble Sort).

La Modularidad usa cuatro archivos en el paquete Ejercicio072\_KevinLainez:

1. Main.java: Controla el flujo principal, crea el arreglo y coordina la llamada a las funciones de entrada, ordenamiento y salida.
2. Menus.java: Gestiona las opciones de continuidad.
3. Operaciones.java: Contiene la función ordenamientoBurbuja(int[]), que implementa la lógica de los ciclos anidados para el intercambio, y una función auxiliar para la lectura/validación de los 10 números.
4. Utilidades.java: Formato estético (cuadros, sello, centrado), presentación del arreglo original y del arreglo ordenado.

El Paradigma es Estructurado/Funcional, centrado en la manipulación de arreglos.

## SUBRUTINAS Y FLUJO PRINCIPAL

Constantes:

- ANCHO = 70.
- TAMANO\_ARREGLO = 10.

Subrutinas Clave (Funcionalidad):

- Operaciones:
  - leerNumeros(Scanner): Solicita y valida los 10 enteros, retornando el arreglo.
  - ordenamientoBurbuja(arreglo): Ordena el arreglo in-place (intercambia elementos).

- Utilidades: mostrarArreglo (imprime el contenido del arreglo antes y después del ordenamiento).

Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
2. Llama a Operaciones.leerNumeros para crear y llenar el arreglo de 10 posiciones.
3. Llama a Utilidades.mostrarArreglo para mostrar el arreglo Original.
4. Llama a Operaciones.ordenamientoBurbuja para ordenar los elementos.
5. Llama a Utilidades.mostrarArreglo para mostrar el arreglo Ordenado.
6. Muestra el menú de continuidad.
7. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el Ejercicio#72: El ordenamiento por intercambio es generalmente considerado el algoritmo de ordenamiento más simple. Escriba una aplicación que implemente el ordenamiento por intercambio. La aplicación debe recibir como entrada 10 enteros. Después de leer los 10 enteros, la aplicación debe ordenarlos en forma ascendente y mostrar el arreglo ordenado (Ejercicio 7.15, Libro 1). Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del programa

```
|BIENVENIDO AL EJERCICIO #72!
CÁLCULO DEL ÁREA DE UN CÍRCULO

INGRESO DEL RADIO
Ingrese el radio del círculo (debe ser > 0):
>> Ingrese valor: 2

--- RESULTADO DEL CÁLCULO ---
Radio ingresado: 2.00
Área del Circulo (π*r²): 12.5664

SELO PERSONALIZADO
Kevin Lainez | klainezs@unah.hn | 072 | 2025-11-03

--- Programa finalizado con éxito ---

Proceso terminado con código de salida 0|
```

## EJERCICIO#73

### Enunciado

Pan comido: Escriba un programa completo en JAVA con las dos funciones alternativas que se especifican a continuación, de las cuales cada una simplemente triplica la variable cuenta definida en main. Después compare y contraste ambos métodos. Estas dos funciones son:

- A. la función triplicarPorValor, que pasa una copia de cuenta por valor, triplica la copia y devuelve el nuevo valor, y
- B. la función triplicarPorReferencia, que pasa cuenta por referencia a través de un parámetro por referencia y triplica el valor original de cuenta a través de su alias (es decir, el parámetro por referencia).

### Análisis

El código del Ejercicio 073 (Pan Comido: Triplicar por Valor vs. Referencia) se centra en la demostración fundamental del mecanismo de paso de argumentos en Java y el contraste de sus efectos, cumpliendo con los requisitos de modularidad y validación.

#### 1. Demostración y Lógica Central

El ejercicio aborda un concepto de Programación Estructurada (PE) típicamente asociado a lenguajes como C/C++ pero crucial para entender la inmutabilidad de primitivos en Java:

- triplicarPorValor(int cuenta): Esta función opera con el tipo primitivo int. Al recibir cuenta, Java pasa una copia del valor. La función triplica esta copia, y el valor original en main permanece sin cambios. Para que el main obtenga el resultado, la función debe devolver el nuevo valor.
- triplicarPorReferencia(ContenedorEntero cuentaRef): Esta función utiliza la clase auxiliar ContenedorEntero (un objeto mutable) para simular el paso por referencia. Aunque Java pasa una copia de la referencia (un "paso por valor de la referencia"), esta copia apunta a la misma ubicación de memoria que el objeto original. Al modificar el campo valor dentro del objeto (cuentaRef.valor \*= 3), el valor original en el main es alterado. La función no necesita devolver nada.

#### 2. Estructura y Modularidad

- ContenedorEntero (Clase Auxiliar): Necesaria para simular la mutabilidad del paso por referencia en Java, manteniendo la responsabilidad del dato mutable.
- Ejercicio073\_Funciones: Aísla la lógica de triplicado en sus dos formas, garantizando la pureza funcional del procesamiento.

- Ejercicio073\_Utilidades: Gestiona el I/O, la validación (entero positivo), la internacionalización y los menús, manteniendo los estándares de la plantilla (sello con fecha 2025-10-16).
- Ejercicio073\_Main: Orquesta la demostración, primero usando el valor primitivo (donde la variable cuentaPorValor no cambia) y luego usando el objeto (cuentaPorReferencia.valor sí cambia), ilustrando claramente el contraste solicitado. El main mantiene el ciclo de repetición del programa.

El código cumple todos los requisitos, utilizando funciones y ciclos de manera correcta para presentar una lección fundamental de la PE sobre el paso de argumentos.

## PROMPT

Creame un código del Ejercicio 073 (Pan Comido) un programa en Java dentro del paquete Ejercicio073 que demuestre y compare dos métodos para triplicar un valor entero inicial, ilustrando el contraste entre el paso por valor (primitivo) y la simulación del paso por referencia (objetos mutables). La estructura debe ser de Programación Estructurada (PE), dividida en:

1. Ejercicio073\_Funciones: Debe contener dos funciones:
  - triplicarPorValor(int cuenta): Recibe y opera sobre una copia del valor primitivo, triplicándolo y devolviendo el resultado. (El original en main debe permanecer sin cambios).
  - triplicarPorReferencia(ContenedorEntero cuentaRef): Recibe una instancia de una clase contenedora mutable (ContenedorEntero) y modifica el valor original dentro del objeto a través de la referencia. (No debe devolver valor).
2. ContenedorEntero: Una clase auxiliar simple para encapsular el int y permitir la simulación del paso por referencia en Java.
3. Ejercicio073\_Utilidades: Gestiona la validación de la entrada (entero positivo), el sello con fecha 2025-10-16, y los menús bilingües/ejecución.
4. Ejercicio073\_Main: Debe orquestar la demostración completa, mostrando el valor inicial, el resultado de triplicarPorValor (y probando que el original no cambió), y el valor de la cuenta después de triplicarPorReferencia (y probando que el original sí cambió). Se deben mantener los menús de idioma y ejecución y la validación de entrada con manejo de InputMismatchException.

## PROMPT GENERADO

Entiendo que deseas un programa modular para el **Ejercicio 073 (Pan comido)** en Java, dentro del paquete `Ejercicio073`, que implemente dos funciones para triplicar una variable entera (`cuenta : triplicarPorValor` (pasa una copia, triplica y devuelve el nuevo valor) y `triplicarPorReferencia` (pasa por referencia y triplica el valor original). El programa debe incluir menús bilingües (español/inglés), un menú de ejecución, validación de entrada con manejo de `InputMismatchException`, y un sello personalizado "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 073 | 2025-10-16". La estructura usará Programación Estructurada (PE) con tres clases: `Ejercicio073_Main` (coordina el flujo), `Ejercicio073_Utilidades` (maneja sello, menús y validación), y `Ejercicio073_Funciones` (contiene las funciones de triplicación). También se incluirá un análisis comparativo. Razonamiento más profundo a no ser por referencia para tipos primitivos, usarán sin comentarios en el código, siguiendo el estilo de tus

How can Grok help? Auto

```

package Ejercicio073;

import java.util.Scanner;

/**
 * Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 073 | 2025-10-16
 */
public class Ejercicio073_Main {

    private static final int MSG_CUENTA = 11;
    private static final int MSG_FINALIZACION = 14;

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

        Ejercicio073_Utilidades.mostrarSelloCorto();

        Ejercicio073_Utilidades.seleccionarIdioma(scanner);

        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio073_Utilidades.mostrarMenuEjecucion(scanner)) {
                ejecutar = false;
                break;
            }
        }

        Ejercicio073_Utilidades.mostrarEncabezado();

        int cuenta = Ejercicio073_Utilidades.leerEnteroPositivo(scanner,
            Ejercicio073_Utilidades.obtenerTexto(MSG_CUENTA));
    }
}

```

## Salida de la ejecución del programa:

```

Project ▾ Run Ejercicio073_Main × ContenedorEntero.java Ejercicio073_Funciones.java Ejercicio073_Utilidades.java Ejercicio073_Main.java

Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 073 | 2025-10-16

*** BIENVENIDO ***
Seleccione el idioma:
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>1

--- Comparación: Triplicar por Valor vs. Referencia ---
Ingrese un valor inicial (entero positivo): 5

--- DEMOSTRACIÓN POR VALOR ---
Valor inicial de cuenta: 5
Resultado de triplicarPorValor: 15
Valor de cuenta después de triplicarPorValor: 5

--- DEMOSTRACIÓN POR REFERENCIA (SIMULADA) ---
Valor de cuenta antes de triplicarPorReferencia: 5
Valor de cuenta después de triplicarPorReferencia: 15

```

## EJERCICIO#74

**Enunciado:** Elevation - U2: Hay que hacer un programa que involucre dos funciones para elevar al cuadrado y para elevar al cubo sin embargo estas funciones tienen que llevar el mismo nombre la única diferencia en cada par es que una hace uso de paso de parámetros por valor y la otra lo hace por referencia. El programa tendrá dos funciones llamadas elevarCd (para elevar al cuadrado) y elevarCb (para elevar al cubo) entonces, al arrancar y luego de dar la bienvenida al usuario presentará una tabla con cinco columnas en la primera aparecerán los números del cero (0) al diez (10), en la segunda el cuadrado de estos números obtenidos por valor, en la tercera el cuadrado obtenido por referencia, en la cuarta el cubo obtenidos por valor y en la última el cubo obtenidos por referencia. (si se puede tener funciones con el mismo nombre en el mismo archivo cpp solo hay que saber cómo hacerlo)

### Análisis/Requisitos

- **Objetivo**
  - Definir el propósito principal del ejercicio.
  - ~~X~~ "Desarrollar un programa modular que demuestre sobrecarga de funciones para calcular cuadrado y cubo por valor y referencia, mostrando una tabla con números del 0 al 10 y sus elevaciones en un cuadro elegante."
- **Reglas/Condiciones**
  - Usar sobrecarga para elevarCd (cuadrado) y elevarCb (cubo).
  - Una versión por valor (int) y otra por referencia (Integer).
  - Mostrar tabla con 5 columnas: N, Cuadrado Valor, Cuadrado Ref, Cubo Valor, Cubo Ref.
  - No hay entrada manual, datos fijos del 0 al 10.
- **Entrada/Salida**
  - Entrada:
    - Ninguna manual, datos predefinidos (0-10).
  - Salida:
    - Bienvenida en cuadro elegante.
    - Tabla con elevaciones en cuadro de 70 caracteres, columnas alineadas.
    - Soporte para repetir el programa.
- **Estructura**
  - Main.java: Flujo principal, genera tabla.
  - Menús.java: Gestión de idioma y continuación.
  - Operaciones.java: Funciones sobrecargadas para elevaciones.
  - Utilidades.java: Formato de cuadros, sello y textos.

- **Estética**
  - ✓ Ancho de consola de 70 caracteres.
  - ✓ Bordes Unicode (Γ, Λ, Τ, Σ, –, |) para diseño simétrico.
  - → Texto centrado en títulos para consistencia visual.
- **Funcionalidades**
  - elevarCd(int) y elevarCd(Integer): Sobrecarga para cuadrado.
  - elevarCb(int) y elevarCb(Integer): Sobrecarga para cubo.
  - mostrarBienvenida: Cuadro de bienvenida.
  - mostrarTabla: Tabla con resultados.
  - seleccionarIdioma y preguntarContinuar: Interactividad.
- **Restricciones**
  - ✓ Números del 0 al 10.
  - ✓ Sin arreglos, uso de ciclos para tabla.
  - ✗ Referencia en Java es por valor de objeto.
- **Soporte**
  - ✓ Soporte multilingüe (Español/Inglés) con tildes.
  - ✓ Opción de repetir el programa.
  - → Validación implícita en datos fijos.
- **Validación y Control**
  - ✓ Lógica asegura tabla de 0 a 10.
  - ✗ No se valida entrada manual por datos predefinidos.

**PROMPT:** Agente nuevo ejercicio el #74 siempre implementa los requerimientos de los ejercicios anteriores con una salida elegante en cada una de las salidas al usuario, el ejercicio consta de lo siguiente:

Hay que hacer un programa que involucre dos funciones para elevar al cuadrado y para elevar al cubo sin embargo estas funciones tienen que llevar el mismo nombre la única diferencia en cada par es que una hace uso de paso de parámetros por valor y la otra lo hace por referencia. El programa tendrá dos funciones llamadas elevarCd (para elevar al cuadrado) y elevarCb (para elevar al cubo) entonces, al arrancar y luego de dar la bienvenida al usuario presentará una tabla con cinco columnas en la primera aparecerán los números del cero (0) al diez (10), en la segunda el cuadrado de estos números obtenidos por valor, en la tercera el cuadrado obtenido por referencia, en la cuarta el cubo obtenidos por valor y en la última el cubo obtenidos por referencia. (si se puede tener funciones con el mismo nombre en el mismo archivo cpp solo hay que saber cómo hacerlo)

Todo el código dividido en nuestro sistemas de 4 archivos, valida entradas, que las salida al usuario sean fáciles de interpretar

**PROMPT GENERADO:**

The screenshot shows a Java code editor interface. On the left, there is a sidebar with file navigation and search functions. The main area displays a Java program named `Ejercicio074`. The code includes a header with a personalized stamp and a menu selection logic. The terminal output at the bottom shows the program's execution and a welcome message.

```

/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 074 | 2025-10-18
 */
package Ejercicio074;
import java.util.Scanner;

public class Menús {
    private static final int ANCHO = 70;
    private static final char BORDE_IZQ = '|';
    private static final char BORDE_DER = '|';
    private static final char LINEA_H = '-';

    public static String seleccionarIdioma(Scanner scanner) {
        int opcion;
        do {
            System.out.println("\n" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
            System.out.println(BORDE_IZQ + centrarTexto("SELECCIÓN DE IDIOMA / LANGUAGE SELECTION",
                ANCHO - 2) + BORDE_DER);
            System.out.println("|" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "|");
            System.out.println(BORDE_IZQ + centrarTexto("1. Español", ANCHO - 2) + BORDE_DER);
            System.out.println(BORDE_IZQ + centrarTexto("2. English", ANCHO - 2) + BORDE_DER);
            System.out.println("[" + String.valueOf(LINEA_H).repeat(ANCHO - 2) + "]");

            System.out.print("Seleccione su idioma / Select language: ");
            if (scanner.hasNextInt()) {
                opcion = scanner.nextInt();
                scanner.nextLine();
            }
            if (opcion == 1 || opcion == 2) {
                break;
            }
        } while (true);
        return opcion == 1 ? "selección de idioma / Selected language: 1" : "selección de idioma / Selected language: 2";
    }
}

```

## Salida de la Ejecución del programa:

The screenshot shows a terminal window within a code editor. It displays the execution of the `Ejercicio074.Main` program. The output shows a welcome message and a table of elevations (Cuadrado, Cubo) from 0 to 10. The user is prompted to continue, and the process exits with code 0.

```

Ejercicio074.Main
¡Bienvenido al programa de elevaciones!

Tabla de Elevaciones
N Cuadrado Valor Cuadrado Ref Cubo Valor Cubo Ref
0 0 0 0 0
1 1 1 1 1
2 4 2 8 2
3 9 3 27 3
4 16 4 64 4
5 25 5 125 5
6 36 6 216 6
7 49 7 343 7
8 64 8 512 8
9 81 9 729 9
10 100 10 1000 10

?Desea continuar? (S/N): n
?Desea continuar? (S/N): N

Process finished with exit code 0

```

## EJERCICIO#75

### Enunciado

Trigonometría básica: Usando la librería math.h haga un programa que en forma tabular muestre los grados desde 0o a 360o y a la par su seno, coseno y tangente. Debe satisfacer las siguientes restricciones:

- a. La tabla debe estar correctamente construida con columnas de un ancho de 15.
- b. Deben usarse solamente tres cifras decimales en los resultados mostrados.
- c. Como saldrán 361 resultados el programa debe hacer una pausa y solicitar que se presione cualquier tecla cada vez que haya mostrado 50 valores.
- d. Despues de cada pausa los encabezados de las columnas deben volver a aparecer.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 075 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Mostrar una tabla de grados de 0 a 360.  
Calcular seno, coseno y tangente usando Math.sin(), Math.cos(), Math.tan().  
Mostrar resultados con 3 cifras decimales.  
Cada columna debe tener ancho de 15 caracteres.  
La tabla debe pausar cada 50 filas y solicitar al usuario presionar una tecla.  
Después de cada pausa, volver a mostrar los encabezados de columnas.  
Validar las entradas si se requiere repetir la operación.  
Incluir menú de idioma (ES / EN) y menú de iniciar/finalizar.  
Mantener la estética uniforme con bordes Unicode.  
Mostrar el sello corto antes del pie final.
- **Modularidad:** Cuatro archivos en paquete Ejercicio075:
  - Ejercicio075\_Main.java → Control del flujo principal, bucle principal y llamada a operaciones.
  - Ejercicio075\_Menus.java → Menús de selección de idioma y menú iniciar/finalizar.
  - Ejercicio075\_Utilidades.java → Funciones gráficas: encabezado, pie, sello corto, pausa y formato estético.
  - Ejercicio075\_Operaciones.java → Función que construye la tabla de grados y calcula seno, coseno y tangente con pausa.
- **Paquete:** Ejercicio075.
- **Estética:** Bordes Unicode (𝔽, ||, ℒ, =) para encabezados y pies.

Encabezados y pies tabulados.  
Sello corto antes del pie final.  
Salida en tabla con columnas de 15 caracteres y 3 cifras decimales.  
Pausas cada 50 filas, reimpresión de encabezados.

- **Paradigma:** Programación estructurada.  
Modularidad en 4 archivos.  
Funciones para cálculos trigonométricos y formato de salida.  
Bucles controlados para flujo de ejecución y pausa de tabla.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio075\_Menu):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio075\_Utilidades):**

- mostrarEncabezado(String titulo) Muestra cabecera decorativa.
- mostrarPie(String msg) Muestra pie decorativo.
- mostrarSelloCorto() Imprime el sello corto.
- pausarTabla() Realiza pausa para la tabla cada 50 filas y espera que el usuario presione una tecla.

#### **Subrutinas (en Ejercicio075\_Operaciones):**

- mostrarTablaTrigonometria() Construye la tabla de 0 a 360 grados con seno, coseno y tangente. Controla pausas cada 50 filas.
- calcularSeno(double grados) Devuelve seno del ángulo en radianes.
- calcularCoseno(double grados) Devuelve coseno del ángulo en radianes.
- calcularTangente(double grados) Devuelve tangente del ángulo en radianes.

#### **Flujo Principal (en Ejercicio075\_Main):**

- main()
- Selección de idioma, bucle principal con menú iniciar/finalizar, ejecución de mostrarTablaTrigonometria().

#### **PROMPT**

Agente vamos con el siguiente ejercicio, respetando el orden y estructura que venimos trabajando. El siguiente ejercicio es el 75 y su enunciado es: Trigonometría básica: Usando la librería math.h haga un programa que en forma tabular muestre los grados desde 0 a 360 y a la par su seno, coseno y tangente. Debe satisfacer las siguientes restricciones:

- a. La tabla debe estar correctamente construida con columnas de un ancho de 15.
- b. Deben usarse solamente tres cifras decimales en los resultados mostrados.

- c. Como saldrán 361 resultados el programa debe hacer una pausa y solicitar que se presione cualquier tecla cada vez que haya mostrado 50 valores.
- d. Despues de cada pausa los encabezados de las columnas deben volver a aparecer.

## PROMPT GENERADO

¡Perfecto! 😊 Vamos a generar los archivos y el informe del Ejercicio 75, respetando exactamente la misma estructura que hemos venido usando: 4 archivos (Main, Menus, Utilidades, Operaciones), menú de idioma, menú de iniciar/finalizar, sello corto y estética uniforme con bordes Unicode.

---

**Informe del Ejercicio 75 — Trigonometría Básica**

---

**Sello personalizado corto**

yaml
 Copiar código

Dany Noguera | Ejercicio075 | 2025-11-08

---

**Requerimientos del programa**

1. Mostrar una tabla de grados de 0 a 360.
2. Calcular seno, coseno y tangente usando `Math.sin()`, `Math.cos()`, `Math.tan()`.

+ Pregunta lo que quieras

0

## Salida de la ejecución del programa:

```

Run - Guia_de_ejercicios
Run Ejercicio075_Main x
G S : 
↑ MENÚ PRINCIPAL ↓
↳ 1. Iniciar programa
↳ 2. Finalizar
Select one option: 1
TABLA TRIGONOMÉTRICA
Grados Seno Coseno Tangente
----- 
0 0.000 1.000 0.000
1 0.017 1.000 0.017
2 0.035 0.999 0.035
3 0.052 0.999 0.052
4 0.070 0.998 0.070
5 0.087 0.996 0.087
6 0.105 0.995 0.105
7 0.122 0.993 0.123
8 0.139 0.990 0.141
9 0.156 0.988 0.158
10 0.174 0.985 0.176
11 0.191 0.982 0.194
12 0.208 0.978 0.213
13 0.225 0.974 0.231
14 0.242 0.970 0.249
15 0.259 0.966 0.268
16 0.276 0.961 0.287

```

## EJERCICIO#76

### Enunciado

Círculo de nuevo: Escriba un programa en JAVA que pida al usuario el radio de un círculo y después llame a la función inline areaCirculo para calcular el área de ese círculo (Ejercicio 6.55, libro 1).

### ANÁLISIS/RESUMEN DEL EJERCICIO 76

#### Sello y Documentación

- **Sello:** VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 076 | Fecha: 2025-11-03) en los cuatro archivos y en un cuadro final.
- **Prompt:** Completo en **Main.java**.
- **Requisitos:** El programa debe ejecutarse varias veces con opción de continuar/salir, y todas las entradas deben ser validadas.

#### Programa y Modularidad

El **Programa** solicita **dos ángulos** ( $A_1$  y  $A_2$ ) y debe aplicar una **validación de orden y rango** compleja. Finalmente, imprime una tabla comparativa de sus funciones trigonométricas.

1. **Validación de Rango:** Ambos ángulos deben estar en  $[0^\circ, 360^\circ]$ .
2. **Validación de Orden:**  $A_2$  debe ser mayor que  $A_1$ . Si no lo es, se debe preguntar al usuario si desea **intercambiar** los valores ( $A_1 \leftrightarrow A_2$ ) o **volver a leer** ambos ángulos.

La **Modularidad** usa **cuatro archivos** en el paquete Ejercicio076\_KevinLainez:

1. **Main.java:** Controla el ciclo de repetición y coordina el flujo de entrada/validación y la impresión de la tabla.
2. **Menus.java:** Gestiona el menú/opciones de continuidad y las opciones de **intercambio** durante la validación.
3. **Operaciones.java:** Contiene todas las **funciones trigonométricas** (seno, coseno, tangente, y sus inversas) utilizando la librería Math de Java.

4. **Utilidades.java**: Formato estético, cuadros, y la función central **gestionarAngulos** que encapsula la lógica de validación, rango, orden e intercambio.

## SUBRUTINAS Y FLUJO PRINCIPAL

### Constantes:

- **ANCHO = 70, MIN\_ANGULO = 0, MAX\_ANGULO = 360.**

### Subrutinas Clave (Funcionalidad):

- **Operaciones**: seno, coseno, tangente, secante, cosecante, cotangente (calculan los valores trigonométricos).
- **Utilidades**: gestionarAngulos (función principal de validación y gestión de intercambio), imprimirTablaTrigonometrica (presenta la comparación de resultados).

### Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
2. Llama a **Utilidades.gestionarAngulos** para solicitar, validar el rango y asegurar el orden ( $A1 < A2$ ), incluyendo la gestión del intercambio.
3. Llama a **Operaciones.imprimirTablaTrigonometrica** para mostrar los resultados de las seis funciones para ambos ángulos.
4. Muestra el menú de continuidad.
5. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el **Ejercicio#76**: Usando la librería math.h haga un programa que solicite al usuario dos ángulos con las siguientes restricciones: a. Ambos ángulos deben estar entre 0 y 360. b. El segundo ángulo debe ser mayor que el primero si el usuario ingresa un número menor debe preguntarle si desea cambiar el segundo por el primero y dejar el primero por el segundo. Es decir si angulo1 = 120 y angulo2 = 60 el sistema debe preguntar “¿Desea que el primer ángulo sea 60 y el segundo 120 (1= si, 2 = volver a leer ambos ángulos? : “ y bueno, actuar conforme lo que decida el usuario en cuanto a los valores de los ángulos. c. Imprimir una tabla que satisfaga todos los requerimientos de la tabla usada en el ejercicio “Trigonometría básica”. Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del programa

```
;BIENVENIDO AL EJERCICIO #76!
VALIDACIÓN DE ÁNGULOS Y TRIGONOMETRÍA

Restricción: 0° ≤ Ángulo 1 < Ángulo 2 ≤ 360°

INGRESO Y VALIDACIÓN DE ÁNGULOS

Ingrese Primer Ángulo [0-360]:
>> Ángulo: 160

Ingrese Segundo Ángulo [0-360]:
>> Ángulo: 200

--- TABLA DE VALORES TRIGONOMÉTRICOS ---

  FUNCIÓN      |    160.00° |    200.00°
  Seno          |     0.3420 |   -0.3420
  Coseno        |    -0.9397 |   -0.9397
  Tangente      |    -0.3640 |    0.3640
  Cosecante     |     2.9238 |   -2.9238
  Secante       |    -1.0642 |   -1.0642
  Cotangente    |    -2.7475 |    2.7475
```

```
--- TABLA DE VALORES TRIGONOMÉTRICOS ---

  FUNCIÓN      |    160.00° |    200.00°
  Seno          |     0.3420 |   -0.3420
  Coseno        |    -0.9397 |   -0.9397
  Tangente      |    -0.3640 |    0.3640
  Cosecante     |     2.9238 |   -2.9238
  Secante       |    -1.0642 |   -1.0642
  Cotangente    |    -2.7475 |    2.7475
```

```
SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 076 | 2025-11-03
```

```
--- Programa finalizado con éxito ---

Proceso terminado con código de salida 0
```

## EJERCICIO#77

### Enunciado

Potenciando: Vamos a crear un programa que compare nuestras capacidades con las de otros programadores para ello vamos a crear una función que permita elevar un número a cualquier potencia recuerde que

- A. Usted debe saber como calcular potencias.
- B. Un número elevado a un número negativo también tiene potencia.
- C. Todo número elevado a la cero es igual a 1 sin excepción.

Una vez creada la función vamos a pedir al usuario que ingrese dos números con las siguientes restricciones:

- A. Ambos ángulos deben estar entre -50 y 50.
- B. El segundo número debe ser mayor que el primero si el usuario ingresa un número menor debe preguntarle si desea cambiar el segundo por el primero y dejar el primero por el segundo. Es decir si num1= 15 y num2 = -15 el sistema debe preguntar “¿Desea que el primer número sea -15 y el segundo 15 (1= si, 2 = volver a leer ambos números? : “ y bueno, actuar conforme lo que decida el usuario en cuanto a los valores de los números.

Una vez leídos los números se pedirán tres exponentes (todos entre -10 y 10) y luego el programa debe mostrar la siguiente tabla:

Nu m	Potencia P1	pow(Num, P1)	Potencia P2	Pow(Num, P2)	Potencia P3	pow(Num, P3)
---------	----------------	-----------------	----------------	-----------------	----------------	-----------------

Donde Potencia P1, Potencia P2 y Potencia P3 son potencias obtenidas con la función que usted creó y Pow(Num, P1), Pow(Num, P2) y Pow(Num, P3) son potencias obtenidas usando la función pow de la biblioteca math.h (en java de la clase math). Esta tabla debe satisfacer las siguientes características:

- A. La tabla debe estar correctamente construida con columnas de un ancho de 15.
- B. Deben usarse solamente tres cifras decimales en los resultados mostrados.
- C. La primera columna está formada por todos los números entre el primero y el segundo número incluyéndose.

### Análisis

El análisis del Ejercicio 077 (Potenciando) confirma una implementación exitosa basada en la Programación Estructurada (PE), destacando la modularidad de las funciones y la complejidad del control de flujo y la presentación de datos. La función central calcularPotencia encapsula toda la lógica aritmética, manejando correctamente los casos de exponente cero y negativo mediante un ciclo for y la aritmética de inversión. La clase Ejercicio077\_Utilidades asume la responsabilidad crucial de la validación estricta de las bases (entre \$-50\$ y \$50\$) y los exponentes (entre \$-10\$ y \$10\$), e implementa la lógica condicional compleja en leerYOrdenarBases que pregunta al usuario si desea intercambiar las bases si \$Num1 > Num2\$. Finalmente, esta misma clase utiliza un ciclo for para iterar sobre el rango de bases y genera una tabla comparativa formatada con ancho de 15 y tres decimales (String.format), cotejando los resultados de la función personalizada contra la función estándar Math.pow, cumpliendo así con todos los requisitos del enunciado.

## PROMPT

Creame un código del Ejercicio 077 (Potenciando) un programa en Java dentro del paquete Ejercicio077 que compare una función de potenciación personalizada con Math.pow. La solución debe incluir: 1) Una función modular calcularPotencia(double base, int exponente) que implemente correctamente la potenciación, incluyendo los casos de exponente cero ( $\$base^0 = 1$ ) y negativo ( $\$base^{-n} = 1/base^n$ ), usando un ciclo for para la multiplicación. 2) La clase Ejercicio077\_Utilidades debe manejar la validación estricta de los números base (entre \$-50\$ y \$50\$) y exponentes (entre \$-10\$ y \$10\$), e implementar la lógica condicional compleja en leerYOrdenarBases para garantizar que el  $\$Num1 \leq Num2$  (mediante la opción de intercambio o relectura). 3) El programa debe generar una tabla comparativa para todos los números en el rango  $[\$Num1, Num2]$  con tres exponentes (\$P1, P2, P3\$), usando String.format con un ancho de 15 y tres cifras decimales para mostrar los resultados de la función personalizada contra Math.pow. La estructura debe ser de Programación Estructurada (PE), con el sello actualizado a 2025-10-16, y manteniendo los menús de idioma y ejecución con validación de entrada.

## PROMPT GENERADO

Entiendo que deseas un programa modular para el **Ejercicio (Potenciando)** en Java, dentro del paquete **Ejercicio078**, que implemente una función para calcular potencias personalizada (`calcularPotencia`), compare sus resultados con `Math.pow`, y genere una tabla con resultados para números en un rango dado, siguiendo las restricciones especificadas. El programa usará Programación Estructurada (PE) con tres clases: **Ejercicio078\_Main** (coordina el flujo), **Ejercicio078\_Utilidades** (maneja sello, menús, validación, y tabla), y **Ejercicio078\_Funciones** (contiene la función de potencia). Incluirá menús bilingües (español/inglés), un menú de ejecución, validación de entrada con `InputMismatchException`, y el sello personalizado "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 078 | 2025-10-16". La tabla mostrará números y sus potencias.

```

package Ejercicio078;
import java.util.Scanner;

/*
 * Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 078 | 2025-10-16
 */
public class Ejercicio078_Main {

    private static final int MSG_NUM1 = 11;
    private static final int MSG_NUM2 = 12;
    private static final int MSG_POTENCIA = 13;
    private static final int MSG_FINALIZACION = 14;
    private static final int MSG_CAMBIO = 15;

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

        Ejercicio078_Utilidades.mostrarSelloCorto();

        Ejercicio078_Utilidades.seleccionarIdioma(scanner);

        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio078_Utilidades.mostrarMenuEjecucion(scanner)) {
                ejecutar = false;
                break;
            }
        }

        Ejercicio078_Utilidades.mostrarEncabezado();
    }
}

```

## Salida de la ejecución del programa:

```

*** BIENVENIDO ***
Seleccione el idioma:
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Ejecutar Programa
[2] Finalizar Programa
>1

--- Generación de Tabla de Potencias (Función vs. Math.pow) ---
Ingrese el número base inicial (entre -50 y 50): 1
Ingrese el número base final (entre -50 y 50): 3
Ingrese el Exponente 1 (entero entre -10 y 10): 2
Ingrese el Exponente 2 (entero entre -10 y 10): -1
Ingrese el Exponente 3 (entero entre -10 y 10): 0

Tabla de Potencias:
Num |Potencia 2 |Math.pow(2) |Potencia -1 |Math.pow(-1) |Potencia 0 |Math.pow(0)
---|---|---|---|---|---|---
1.00 |1.000 |1.000 |1.000 |1.000 |1.000 |1.000
2.00 |4.000 |4.000 |0.500 |0.500 |1.000 |1.000
3.00 |9.000 |9.000 |0.333 |0.333 |1.000 |1.000

```

## EJERCICIO #78

### **enunciado:**

Modifique el programa del ejercicio "Simple venta" para implementar una librería propia.

### **análisis:**

Este programa implementa una simulación de venta con un enfoque modular, donde la lógica de cálculo de descuentos e impuestos se aísla completamente en una clase separada llamada GestorVenta.java, actuando como una "librería propia". El objetivo principal es solicitar al usuario la cantidad de artículos, el precio unitario y si el cliente es de la tercera edad (entradas), y luego aplicar un conjunto complejo de reglas de descuento e impuesto para calcular el precio final (salida). Las entradas deben ser valores numéricos positivos. La clase GestorVenta contiene los métodos para determinar el porcentaje de descuento (basado en el monto bruto y la edad), calcular el porcentaje de impuesto (basado en el descuento) y devolver todos los resultados. La clase Operaciones se encarga únicamente de la interacción con el usuario y de la orquestación, llamando al GestorVenta. Los casos límite incluyen cantidades/precios no positivos y la validación de la condición de tercera edad. La complejidad es moderada, centrada en la implementación precisa de las reglas condicionales de negocio dentro de la "librería".

### **prompt**

Modifica el programa de "Simple venta" en Java para implementar una clase GestorVenta que actúe como una librería. Esta clase debe contener toda la lógica para calcular el total bruto, determinar el porcentaje de descuento y el porcentaje de impuesto, y devolver todos los resultados de la transacción para su posterior impresión.

### **prompt generado:**

Para cumplir con el requisito de implementar una librería propia, la solución se estructuró en cinco clases, siendo GestorVenta.java el nuevo componente modular. Esta clase centraliza la compleja lógica de negocio, incluyendo un método público calcularVenta que acepta la cantidad, el precio unitario y la condición de tercera edad. Dentro de esta clase, la lógica para calcular el descuento y el impuesto se encapsula en métodos privados, garantizando que las reglas condicionales (como el descuento escalonado y las reglas de impuesto basadas en el descuento) se manejen internamente y sean reutilizables. Por otro lado, la clase Operaciones se encargó de la lectura de las entradas del usuario (validando valores positivos y la opción de tercera edad) y de llamar a una instancia de GestorVenta. De esta manera, Operaciones se limita a la interfaz de usuario, y GestorVenta actúa como un servicio de cálculo, devolviendo todos los resultados en un mapa para su presentación final.

## Salida:

```
Windows PowerShell
1. Simular Venta
2. Finalizar el Programa
Seleccione una opción (1 o 2): 1

--- MÓDULO: CÁLCULO DE VENTA Y LIBRERÍA PROPIA ---
Ingrese la cantidad total de artículos: 4
Ingrese el precio unitario (L.): 15.50

¿El cliente es de la tercera edad?
1. Sí
2. No
Seleccione una opción (1 o 2): 2

--- DETALLE DE LA VENTA ---
-----
Estatus 3ra. Edad: NO
Total Bruto (Subtotal): L. 62.00
Descuento: L. 6.20 ( 10.0%)
Impuesto (IVA): L. 6.70 ( 12.0%)
-----
TOTAL NETO A PAGAR: L. 62.50
-----

--- MENÚ DE CONTROL ---
1. Simular Venta
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

¡Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> |
```

## EJERCICIO#79

**Enunciado:** Modifique el programa del ejercicio “Cripto-funciones” para implementar una librería propia.

### Análisis

El programa sigue un flujo estructurado con las siguientes etapas:

1. **Inicio:** Muestra un sello personal y permite seleccionar el idioma (Español o Inglés) mediante un menú cíclico.
2. **Menú Principal:** Ofrece opciones para codificar o decodificar un número, utilizando ciclos para repetir hasta que el usuario decida salir.
3. **Entrada y Validación:**
  - Sigue la entrada de un número de cuatro dígitos (1000-9999).
  - Valida la entrada con la función validarNumeroCuatroDigitos, reintentando si es inválida.
4. **Procesamiento:**
  - **codificar(int numero):** Suma 7 a cada dígito y aplica módulo 10, luego intercambia el primer con el tercer dígito y el segundo con el cuarto. // Explicación: Transforma el número original en un valor cifrado seguro.
  - **decodificar(int numero):** Invierte el proceso, reordenando dígitos e restando 7 con ajuste por módulo 10. // Explicación: Recupera el número original a partir del cifrado.
  - Estas funciones residen en MiLibreria.java, separando la lógica criptográfica.
5. **Salida:** Muestra resultados en un cuadro elegante de 70 caracteres, con columnas para el número original y transformado.
6. **Ciclo de Repetición:** Usa preguntarContinuar para decidir si el programa continúa, gestionado por un bucle do-while.

Detalle de Funciones en MiLibreria.java

- **matemáticas:** codificar y decodificar realizan los cálculos criptográficos.
- **validaciones:** validarNumeroCuatroDigitos asegura que la entrada sea un número de cuatro dígitos.
- **formato de texto:** centrarTexto alinea el texto en los cuadros, y mostrarSello genera el encabezado personalizado.
- **procesamiento de datos:** procesarOpcion gestiona la lógica de la elección del usuario.
- **selección de idioma:** seleccionarIdioma permite elegir el idioma y ajusta los mensajes.

- **procesamiento:** iniciarPrograma coordina el flujo general con ciclos.

El uso de una librería propia mejora la estructura al centralizar la lógica, facilitar actualizaciones y permitir reutilización en otros proyectos.

### PROMPT:

Agente empecemos con la nueva sección de ejercicios, en lo que hagamos ahora en adelante deberán de tener la nueva funciones propias ciclos propios y librerías propias, junto con los parámetros anteriores ahora nuestro sistema de archivos se actualizara de esta forma:

Ejercicio ###

  └─ Ejercicio###\_Main.java

  └─ MiLibreria.java ← Contiene TODO (ciclos, funciones y librerías propias)

Un ejemplo de las funciones que hemos usado que deberán de ir

Funciones matemáticas

Funciones de procesamiento

Validaciones

Formato de texto

Procesamiento de datos

Selección de idioma de preferencia

Formato de salida elegante como en los demás

Y todas las que creas necesarias, ahora con esto vamos a nuestro ejercicio, que es curioso porque uno depende del otro y ese otro a la vez depende de otro, a continuación desarrollo lo que te digo. Nuestro ejercicio el #79 Modifique el programa del ejercicio “Cripto-funciones” para implementar una librería propia.

Y el ejercicio “Cripto-funciones” es este: Cripto-funciones: Tomaremos como base el ejercicio “Criptografía” de esta guía, vamos ahora a hacer que codificar y decodificar sean funciones.

Ese ejercicio como lo vez depende de otro, que es este:

Criptografía: Una compañía desea transmitir datos a través del teléfono, pero le preocupa que sus teléfonos puedan estar intervenidos. Todos los datos se transmiten como enteros de cuatro dígitos. La compañía le ha pedido a usted que escriba un programa que cifre sus datos, de manera que éstos puedan transmitirse con más seguridad. Su programa debe leer un entero de cuatro dígitos introducido por el usuario y cifrarlo de la siguiente manera: reemplace cada dígito con (el resultado

de sumar 7 al dígito) módulo 10. Luego intercambie el primer dígito con el tercero, e intercambie el segundo dígito con el cuarto. Después imprima el entero cifrado.

Escriba un programa separado que reciba como entrada un entero de cuatro dígitos cifrado, y que lo descifre para formar el número original.

Con todo el contexto dado y tu entrenamiento dame lo que te solicito de manera ordena y elegante

### PROMPT GENERADO:

### Salida de la ejecución del programa:

```
Seleccione su idioma / Select language: 2
Idioma seleccionado / Selected language: English

CRYPTO-FUNCTIONS MENU
1. Encode number
2. Decode number

Seleccione una opción: 1
Enter a 4-digit number: 4500

ENCODING RESULT
Original number    4500
Encoded number    7712

Do you want to continue? (Y/N): n
Do you want to continue? (Y/N): N

Process finished with exit code 0
```

## EJERCICIO#80

### Enunciado

Modifique el programa del ejercicio “Tabla” para implementar una librería propia. Programa Tabla: programa que luego dar la bienvenida al usuario le muestre la siguiente tabla y entonces le pida un número del 1 al 15, tras validar la entrada debe entonces crear una tabla que vaya desde 1 hasta el número que el usuario haya indicado.

Se aclara que la tabla se debe generar con ciclos y no con condicionales anidados, funciones y librerías propias.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 080| 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Implementar una librería propia para manejar funciones de entrada y salida de datos.
  - Mostrar un mensaje de bienvenida al usuario.
  - Solicitar un número entero entre 1 y 15.
  - Validar que la entrada esté dentro del rango (1-15).
  - Generar una tabla de multiplicar del 1 hasta el número ingresado usando ciclos.
  - La tabla debe generarse mediante funciones propias; no se permiten condicionales anidados para la construcción de la tabla.
  - Incluir menú de idioma (ES / EN).
  - Incluir menú iniciar/finalizar con bucle principal para repetir la operación.
  - Mostrar sello corto y pie final al terminar la operación.
  - Mantener consistencia estética con bordes Unicode y formato uniforme con los demás ejercicios.
- **Modularidad:** Cuatro archivos en paquete Ejercicio080:
  - Ejercicio080\_Main.java → Control del flujo principal; contiene el prompt y el bucle principal de ejecución.
  - Ejercicio080\_Menu.java → Manejo de menús: selección de idioma y menú iniciar/finalizar.
  - Ejercicio080\_Utilidades.java → Funciones gráficas: encabezados, pies, sello corto, lectura validada de enteros.
  - Ejercicio080\_Operaciones.java → Función generarTabla(), contiene la lógica para construir la tabla y mostrar resultados.
- **Paquete:** Ejercicio080.

- **Estética:** Uso de bordes Unicode („, „, ||, =) para encabezados y pies.  
Encabezados y pies formateados de manera uniforme en todos los módulos.  
Mensajes claros de bienvenida y resultados.  
Sello corto mostrado antes del pie final para mantener la consistencia estética de ejercicios previos.  
Tablas tabuladas de manera uniforme.
- **Paradigma:** Programación estructurada.  
Uso de funciones y métodos estáticos.  
Control de flujo secuencial mediante bucles y condicionales simples.  
Modularización en 4 archivos para separación de responsabilidades.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio080\_Menu):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio080\_Utilidades):**

- mostrarEncabezado(String titulo)
- Imprime encabezado decorativo con bordes Unicode.
- mostrarPie(String msg)
- Imprime pie decorativo con bordes Unicode.
- mostrarSelloCorto()
- Muestra el sello corto del desarrollador antes del pie final.
- leerEntero(String msg, int min, int max)
- Función propia que solicita un número entero y valida que esté dentro del rango indicado.

#### **Subrutinas (en Ejercicio080\_Operaciones):**

- generarTabla():
- Muestra encabezado y mensaje de bienvenida.
- Llama a leerEntero() para obtener el número límite de la tabla (1-15).
- Genera la tabla de multiplicar utilizando ciclos anidados.
- Muestra la tabla de manera tabulada.
- Llama a mostrarSelloCorto() y mostrarPie() al finalizar la operación.

#### **Flujo Principal (en Ejercicio080\_Main):**

- main():
- Selección de idioma con Ejercicio080\_Menu.seleccionarIdioma().
- Bucle principal con menuInicioSalir() para repetir o finalizar.
- Llamada a Ejercicio080\_Operaciones.generarTabla() si el usuario decide iniciar.

## PROMPT

Agente vamos con el siguiente ejercicio, respetando el orden y estructura que venimos trabajando. El siguiente ejercicio es el 80 y su enunciado es: Modifique el programa del ejercicio “Tabla” para implementar una librería propia. Programa Tabla: programa que luego dar la bienvenida al usuario le muestre la siguiente tabla y entonces le pida un número del 1 al 15, tras validar la entrada debe entonces crear una tabla que vaya desde 1 hasta el número que el usuario haya indicado. Se aclara que la tabla se debe generar con ciclos (no con condicionales anidados), con funciones y librerías propias.

## PROMPT GENERADO

¡Perfecto! 😊 Vamos a generar los 4 archivos completos del Ejercicio 80, respetando **exactamente el mismo formato y estructura que venimos trabajando**: menú de idioma, menú iniciar/finalizar, bucle principal, sello corto, estética uniforme y modularidad en 4 archivos.

---

Ejercicio080\_Main.java

javaCopiar código

```
// 
// | Desarrollador: Dany Noguera (dany.noguera@unah.hn)
// | Proyecto: Ejercicio #80
// | Prompt:
// | "Modificar el programa 'Tabla' para implementar librería propia. Luego de
// | dar la bienvenida al usuario, mostrar tabla y pedir un número entre 1 y 15,
// | validar entrada y crear tabla hasta el número ingresado usando ciclos y
// | funciones propias."
// | Fecha: 2025-11-10
// 
// Versión corta: Dany Noguera | Ejercicio080 | 2025-11-10
```

Salida de la ejecución del programa:

Run - Guia\_de\_ejercicios

Ejercicio080\_Main

1. Español  
2. English  
Seleccione / Select: 1

MENÚ PRINCIPAL

1. Iniciar programa  
2. Finalizar  
Seleccione una opción: 1

TABLA DE MULTIPLICAR

Bienvenido al programa de tablas de multiplicar.  
Ingrese un número entre 1 y 15: 7

1	2	3	4	5	6	7
2	4	6	8	10	12	14
3	6	9	12	15	18	21
4	8	12	16	20	24	28
5	10	15	20	25	30	35
6	12	18	24	30	36	42
7	14	21	28	35	42	49

[Dany Noguera | Ejercicio080]

Proceso finalizado con éxito.

## EJERCICIO#81

### Enunciado

Modifique el programa del ejercicio "Imprimiendo Líneas" para implementar una librería propia

## ANÁLISIS/RESUMEN DEL EJERCICIO 81

### Sello y Documentación

- **Sello:** VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 081 | Fecha: 2025-11-03) en los cuatro archivos y en un cuadro final.
- **Prompt:** Completo en **Main.java**.
- **Propósito:** Reorganizar las funciones de formato y dibujo en un paquete/librería externa para su reutilización.

### Programa y Modularidad (Librería Externa)

El **Programa** requiere modificar el ejercicio "Imprimiendo Líneas" para que la lógica de dibujo y el formato provengan de una librería propia.

La **Modularidad** usa **cuatro archivos** en el paquete **Ejercicio081\_KevinLainez**, pero depende de una **Librería Externa** (**libreria\_KevinLainez**):

1. **Main.java:** Controla el ciclo de repetición y coordina la llamada a la función de dibujo de la Librería.
2. **Menus.java:** Gestiona el menú/opciones de continuidad.
3. **Operaciones.java:** Contiene la lógica principal de la **Librería Externa** (**MiLibreria.java**), agrupando allí todas las funciones de formato, sello, y la función de **dibujarPatronDeLineas**.
4. **Utilidades.java:** Reservado para utilidades específicas del **Main.java** que no sean de formato (ej., manejo de entradas validadas).

**Subrutinas Clave (en la LIBRERÍA MiLibreria dentro de Operaciones.java):**

- **MiLibreria:** `mostrarSaludo`, `mostrarSelloCorto`, `centrarTexto`, `dibujarLineaHorizontal`.
- **MiLibreria:** `dibujarPatronDeLineas()` (Lógica de dibujo del patrón).

### **Flujo Principal (en Main.java):**

1. Inicia ciclo do-while (continuar/salir).
2. Llama directamente a la función **MiLibreria.dibujarPatronDeLineas()** (ubicada en Operaciones.java).
3. Muestra el menú de continuidad.
4. Muestra el sello al finalizar.

### **PROMPT**

"Agente te recuerdo los nuevos parámetros... se presenta el **Ejercicio#81**: Modifique el programa del ejercicio “Imprimiendo líneas” para implementar una librería propia. Con esa información presenta nuestra modularidad de 4 archivos."

### **Salida del programa**

```
¡BIENVENIDO AL EJERCICIO #81!
USANDO LIBRERÍA EXTERNA Y ESTRUCTURA SIMPLE

** PATRÓN DE LÍNEAS USANDO LIBRERÍA **
=
===
=====
=====
=====
=====
=====
=====
=====

SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 081 | 2025-11-03

--- Programa finalizado con éxito ---

Proceso terminado con código de salida 0
```

## EJERCICIO#82

### Enunciado

Modifique el programa del ejercicio “Alinea ésta 60” para implementar una librería propia

### Análisis

El código, empaquetado en Ejercicio082 y estructurado en tres clases principales (Ejercicio082\_Main, Ejercicio082\_Utilidades, Ejercicio082\_Operaciones) junto con una librería propia reutilizable LibreriaFiguras, implementa un programa en Java que dibuja un cuadrado hueco de tamaño variable (1 a 20) usando cualquier carácter ingresado por el usuario, mediante ciclos for anidados y funciones, sin uso de arreglos; Main coordina el flujo con un menú bilingüe de idioma (español/inglés/salir) y un menú principal con opciones de dibujar, volver o salir, permitiendo ejecución múltiple mediante una pregunta de continuar (s/n) al final de cada operación; Utilidades encapsula un sello personalizado centrado en marco de 70 caracteres con "Lis♥", código "082" y fecha 2025-11-04, junto con validación robusta de entrada (lado numérico en rango, carácter único con longitud 1), manejo de excepciones, reintentos automáticos, internacionalización completa, centrado dinámico y limpieza de buffer; Operaciones actúa como interfaz que invoca exclusivamente a LibreriaFiguras.dibujarCuadrado(...) con un mensaje bilingüe previo; y la librería LibreriaFiguras contiene solo la función pura dibujarCuadrado(int, char) con ciclos anidados para bordes e interior, documentada con Javadoc y diseñada para ser reutilizable en otros programas, garantizando un diseño altamente modular, profesional, robusto, extensible, sin dependencias externas ni estructuras prohibidas, y alineado estrictamente con las mejores prácticas de programación orientada a objetos y los requisitos académicos del curso.

### PROMPT

Ahora créame un programa en Java empaquetado en Ejercicio082 y dividido estrictamente en tres clases (Ejercicio082\_Main, Ejercicio082\_Utilidades, Ejercicio082\_Operaciones) más una librería propia llamada LibreriaFiguras en el mismo paquete, que dibuje un cuadrado hueco de tamaño entre 1 y 20 usando cualquier carácter ingresado por el usuario, mediante ciclos for anidados y funciones, sin usar arreglos; el programa debe iniciar con un sello personalizado centrado en marco de 70 caracteres mostrando "SELLO PERSONALIZADO - VERSIÓN CORTA", seguido de un menú bilingüe de idioma (1. Español, 2. English, 3. Salir), y tras elegir idioma, un menú principal con: 1. Dibujar cuadrado, 2. Volver al menú de idioma, 3. Salir; al seleccionar dibujar, solicitar lado (1-20) y un solo carácter de relleno con validación estricta (rango, tipo, longitud 1) y reintentos automáticos; la función public static void dibujarCuadrado(int lado, char caracter) debe estar solo en LibreriaFiguras, con ciclos anidados y Javadoc, y Operaciones debe invocarla con un mensaje bilingüe previo; al finalizar cada dibujo, preguntar ¿Desea continuar? (s/n) con validación bilingüe; Main coordina el flujo con bucles y switch, Utilidades encapsula sello con centrado dinámico, menús, validación robusta, mensajes claros y manejo de excepciones, y LibreriaFiguras debe ser reutilizable, limpia y sin entrada/salida; el código debe ser modular, profesional, robusto, reutilizable, sin arreglos, con funciones bien definidas y cumplir estrictamente con buenas prácticas de POO y restricciones académicas.

## PROMPT GENERADO

Crea un programa en Java empaquetado en `Ejercicio082` y dividido estrictamente en tres clases (`Ejercicio082_Main`, `Ejercicio082_Utilidades`, `Ejercicio082_Operaciones`) que dibuje un cuadrado hueco de tamaño entre 1 y 20 usando **cualquier carácter** ingresado por el usuario, mediante ciclos `for` **anidados y funciones**, sin usar arreglos, y que implemente una **librería propia** llamada `LibreriaFiguras` (en el mismo paquete o subpaquete) que contenga **únicamente** la función `public static void dibujarCuadrado(int lado, char caracter)` responsable de imprimir el cuadrado hueco con bordes del carácter y espacio interno, usando solo ciclos; el programa debe iniciar con un **sello personalizado centrado** en marco de 70 caracteres mostrando "SELLO PERSONALIZADO - VERSIÓN CORTA" y "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 082 | 2025-11-04", seguido de un **menú bilingüe de idioma** (1. Español, 2. English, 3. Salir), y tras elegir idioma, un **menú principal** con: 1. Dibujar cuadrado, 2. Volver al menú de idioma, 3. Salir; al seleccionar dibujar, solicitar **lado (1-20)** y **un solo carácter de relleno con validación estricta** (rango, tipo, longitud 1) y reintentos automáticos; la clase `Ejercicio082_Operaciones` debe actuar como **interfaz** que invoque exclusivamente a `LibreriaFiguras.dibujarCuadrado(...)` pasando los parámetros y un mensaje bilingüe previo; al finalizar cada dibujo, preguntar **¿Desea continuar? (s/n)** con validación bilingüe; `Main` coordina el flujo con bucles y `switch`, `Utilidades` maneja el manejo de excepciones amigable, menús, validación robusta, mensajes claros y manejo de excepciones; solo la llamada a la librería; la

## Salida de la ejecución del programa:



## EJERCICIO#83

**Enunciado:** Modifique el programa del ejercicio “Instrucción asistida por computadora” para implementar una librería propia.

### Análisis

El programa sigue un flujo optimizado con estas etapas:

1. **Inicio:** Muestra un sello personalizado y permite seleccionar el idioma (Español o Inglés) mediante un menú interactivo.
2. **Procesamiento de Respuestas:**
  - Utiliza un ciclo for para registrar 10 respuestas.
  - El estudiante elige entre "Correcta" (1) o "Incorrecta" (2), validado por validarOpcion.
  - Muestra mensajes aleatorios de feedback con mostrarMensaje.
3. **Cálculo y Validación:**
  - calcularPorcentaje determina el porcentaje de respuestas correctas.
  - Si el porcentaje es menor al 75%, se muestra mostrarMensajeAyuda y termina.
4. **Salida:** Presenta resultados en un cuadro elegante con mostrarResultados, usando bordes simétricos de 70 caracteres.
5. **Ciclo de Repetición:** preguntarContinuar permite repetir el proceso con un bucle do-while.

Detalle de Funciones en MiLibrería.java

- **matemáticas:** calcularPorcentaje realiza el cálculo del rendimiento.
- **validaciones:** validarOpcion asegura entradas válidas (1 o 2).
- **formato de texto:** centrarTexto alinea el contenido, y mostrarSello genera el encabezado.
- **procesamiento de datos:** procesarRespuestas gestiona el flujo de las 10 respuestas.
- **selección de idioma:** seleccionarIdioma ajusta los mensajes según la preferencia.
- **procesamiento de mensajes:** obtenerMensajeCorrecto y obtenerMensajeIncorrecto proporcionan feedback aleatorio.

La librería mejora la estructura al centralizar la lógica, facilitando mantenimiento y escalabilidad.

## EJERCICIO#84

### Enunciado

Nos gustan los ángulos rectos: Cree una librería que imprima diversos tipos de cuadrados:

- a. Cuadrado solo con el borde y vacío por dentro.
- b. Cuadrado lleno.
- c. Cuadrado solo la mitad lleno.
- d. Rectángulo vacío.
- e. Rectángulo lleno.
- f. Triángulo (como sea)

Y ahora haga un programa que pregunte al usuario qué desea dibujar y con qué carácter y lo presente en pantalla (el usuario debe poder ingresar la longitud de los lados de los cuadrados y la altura del triángulo).

NOTA: Si elige rectángulo se debe leer lado más largo y lado más corto y asegurarse que el lado más largo sea el doble del lado más corto USANDO PARA ELLO UNA FUNCIÓN.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 084| 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Crear una librería propia que permita dibujar figuras geométricas:
  - a. Cuadrado solo con el borde y vacío por dentro.
  - b. Cuadrado lleno.
  - c. Cuadrado solo la mitad lleno.
  - d. Rectángulo vacío.
  - e. Rectángulo lleno.
  - f. Triángulo (forma libre).
- Permitir al usuario seleccionar la figura a dibujar mediante un menú.
- Solicitar al usuario el carácter a utilizar para dibujar.
- Solicitar al usuario la longitud de los lados del cuadrado o la altura del triángulo.
- Si se selecciona un rectángulo, leer el lado más largo y el lado más corto, asegurando que el lado más largo sea el doble del lado más corto mediante una función.
- Validar todas las entradas: números positivos y consistentes con las restricciones.
- Incluir menú de idioma (ES / EN).
- Incluir menú iniciar/finalizar con bucle principal para permitir repetir la operación.
- Mostrar sello corto y pie final al concluir cada ejecución.

- Mantener estética uniforme con bordes Unicode y formato de los ejercicios previos.
- **Modularidad:** Cuatro archivos en paquete Ejercicio084:
  - Ejercicio084\_Main.java → Control del flujo principal; contiene el prompt y el bucle principal de ejecución.
  - Ejercicio084\_Menus.java → Manejo de menús: selección de idioma y menú iniciar/finalizar.
  - Ejercicio084\_Utilidades.java → Funciones gráficas: encabezados, pies, sello corto, lectura validada de enteros y caracteres.
  - Ejercicio084\_Operaciones.java → Funciones para dibujar figuras geométricas (dibujarCuadradoVacio(), dibujarCuadradoLleno(), dibujarTriangulo(), validarLadosRectangulo(), etc.).
- **Paquete:** Ejercicio084.
- **Estética:** Uso de bordes Unicode („, „, ||, =) para encabezados y pies.  
 Encabezados y pies uniformes en todos los módulos.  
 Sello corto mostrado antes del pie final.  
 Resultados claros y bien tabulados en consola.  
 Menús y prompts amigables para el usuario en ambos idiomas.
- **Paradigma:** Programación estructurada.  
 Uso de funciones y métodos estáticos.  
 Control de flujo secuencial mediante bucles y condicionales.  
 Modularización en 4 archivos para separación de responsabilidades.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio084\_Menus):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio084\_Utilidades):**

- mostrarEncabezado(String titulo)
- Imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg)
- Imprime un pie decorativo con bordes Unicode.
- mostrarSelloCorto()
- Muestra el sello corto antes del pie final.
- leerEntero(String msg, int min, int max)
- Solicita un número entero con validación de rango.
- leerCaracter(String msg)

- Solicita un carácter para dibujar la figura.

### **Subrutinas (en Ejercicio084\_Operaciones):**

- seleccionarFigura():
  - Muestra un menú de figuras disponibles.
  - Llama a la función correspondiente según la selección del usuario.
  - Solicita las dimensiones necesarias (lados, altura) usando leerEntero() y leerCaracter().
  - Si se selecciona un rectángulo, llama a validarLadosRectangulo() para asegurar que el lado más largo sea el doble del lado más corto.
  - Muestra la figura en pantalla usando funciones de dibujo.
  - Llama a mostrarSelloCorto() y mostrarPie() al finalizar la operación.
  - Funciones de dibujo:
    - dibujarCuadradoVacio(int lado, char c)
    - dibujarCuadradoLleno(int lado, char c)
    - dibujarCuadradoMitadLleno(int lado, char c)
    - dibujarRectanguloVacio(int largo, int corto, char c)
    - dibujarRectanguloLleno(int largo, int corto, char c)
    - dibujarTriangulo(int altura, char c)
    - validarLadosRectangulo(int largo, int corto)
  - Devuelve true si el lado largo es el doble del lado corto, false en caso contrario.

### **Flujo Principal (en Ejercicio084\_Main):**

- Selección de idioma con Ejercicio084\_Menus.seleccionarIdioma().
- Bucle principal con menuInicioSalir() para repetir o finalizar.
- Llamada a Ejercicio084\_Operaciones.seleccionarFigura() si el usuario decide iniciar.

### **PROMPT**

Agente vamos con el siguiente ejercicio, respetando el orden y estructura que venimos trabajando y las reglas que te he proporcionado. El siguiente ejercicio es el 84 y su enunciado es: Nos gustan los ángulos rectos: Cree una librería que imprima diversos tipos de cuadrados:

- Cuadrado solo con el borde y vacío por dentro.
- Cuadrado lleno.
- Cuadrado solo la mitad lleno.
- Rectángulo vacío.
- Rectángulo lleno.
- Triángulo (como sea)

Y ahora haga un programa que pregunte al usuario qué desea dibujar y con qué carácter y lo presente en pantalla (el usuario debe poder ingresar la longitud de los lados de los cuadrados y la altura del triángulo).

**NOTA:** Si elige rectángulo se debe leer lado más largo y lado más corto y asegurarse que el lado más largo sea el doble del lado más corto USANDO PARA ELLO UNA FUNCIÓN.

## PROMPT GENERADO

¡Perfecto! 😊 Vamos a preparar el **informe completo del Ejercicio 84**, respetando **exactamente la misma estructura, formato y reglas** que venimos utilizando en todos los ejercicios anteriores.

---

## Informe del Ejercicio 84 — Nos gustan los ángulos rectos

---

### Sello personalizado corto

yaml

Dany Noguera | Ejercicio084 | 2025-11-11

Copiar código

---

### Requerimientos del programa

1. Crear una librería propia que permita dibujar figuras geométricas:
  - a. Cuadrado solo con el borde y vacío por dentro.

Continuar generando

+ Pregunta lo que quieras

0

## **Salida de la ejecución del programa:**

## EJERCICIO#85

### Enunciado

Ficheros y funciones: Cree una librería propia para implementar una función que reciba un arreglo unidireccional y un archivo para que, almacene línea por línea del archivo en cada una de las posiciones del vector. El vector debe ser del tipo string.

## ANÁLISIS/RESUMEN DEL EJERCICIO 85

### Sello y Documentación

- **Sello:** VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 085 | Fecha: 2025-11-03) en los cuatro archivos y en un cuadro final.
- **Prompt:** Completo en **Main.java**.
- **Propósito:** Implementar una función central en la librería propia para manejar la entrada/salida (I/O) de datos.

### Programa y Modularidad (Librería Externa con I/O)

El **Programa** debe probar una función de la librería que lee un **archivo (fichero)** y almacena su contenido **línea por línea** en un **arreglo unidimensional de String**. La prueba debe incluir la creación de un archivo de prueba.

La **Modularidad** usa **cuatro archivos** en el paquete Ejercicio085\_KevinLainez, que importan la **Librería Externa** (MiLibreria):

1. **Main.java:** Controla el flujo principal, crea el archivo de prueba, define el arreglo (String[]), y coordina la llamada a la función de lectura de la librería.
2. **Menus.java:** Gestiona las opciones de continuidad.
3. **Operaciones.java:** Contiene la función clave **leerFicheroAVector(String[] vector, String archivo)** y la función auxiliar para **crearArchivoPrueba** (ambas residen en la librería).
4. **Utilidades.java:** Formato estético y presentación de la salida de la lectura (muestra el contenido del arreglo lleno).

### Subrutinas Clave (en la LIBRERÍA MiLibreria / Operaciones.java):

- **leerFicheroAVector:** Recibe un arreglo (String[]) y la ruta del fichero. Lee y almacena cada línea del fichero en una posición del arreglo, manejando excepciones de I/O y la validación del tamaño del arreglo.
- **crearArchivoPrueba:** Función auxiliar para asegurar que exista un archivo de texto con contenido para la prueba.

## Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
2. Llama a crearArchivoPrueba para generar un archivo temporal.
3. Define un arreglo String[] vacío con capacidad suficiente.
4. Llama a **Operaciones.leerFicheroAVector** para llenar el arreglo.
5. Muestra el contenido final del arreglo (usando Utilidades).
6. Muestra el menú de continuidad.
7. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el **Ejercicio#85**: Cree una librería propia para implementar una función que reciba un arreglo unidireccional y un archivo para que, almacene línea por línea del archivo en cada una de las posiciones del vector. El vector debe ser del tipo string. Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del programa

```
¡BIENVENIDO AL EJERCICIO #85!
LECTURA DE ARCHIVO A ARREGLO (STRING)

PREPARACIÓN: Archivo 'datos_ejercicio85.txt' creado con 5 líneas.
              Vector creado con tamaño: 5

EJECUTANDO: MiLibreria.leerFicheroAVector()...

RESULTADO: Datos leídos del fichero y almacenados en el vector.

          Contenido del Vector:
          Posición [0]: Esta es la linea de prueba número 1
          Posición [1]: Esta es la linea de prueba número 2
          Posición [2]: Esta es la linea de prueba número 3
          Posición [3]: Esta es la linea de prueba número 4
          Posición [4]: Esta es la linea de prueba número 5

SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 085 | 2025-11-03

--- Programa finalizado con éxito ---

Proceso terminado con código de salida 0
```

## EJERCICIO#86

### Enunciado

Imprimiendo en archivo: Haga un programa que escriba un mensaje, escrito por el usuario, en un archivo de texto para ello se debe pedir al usuario el mensaje.

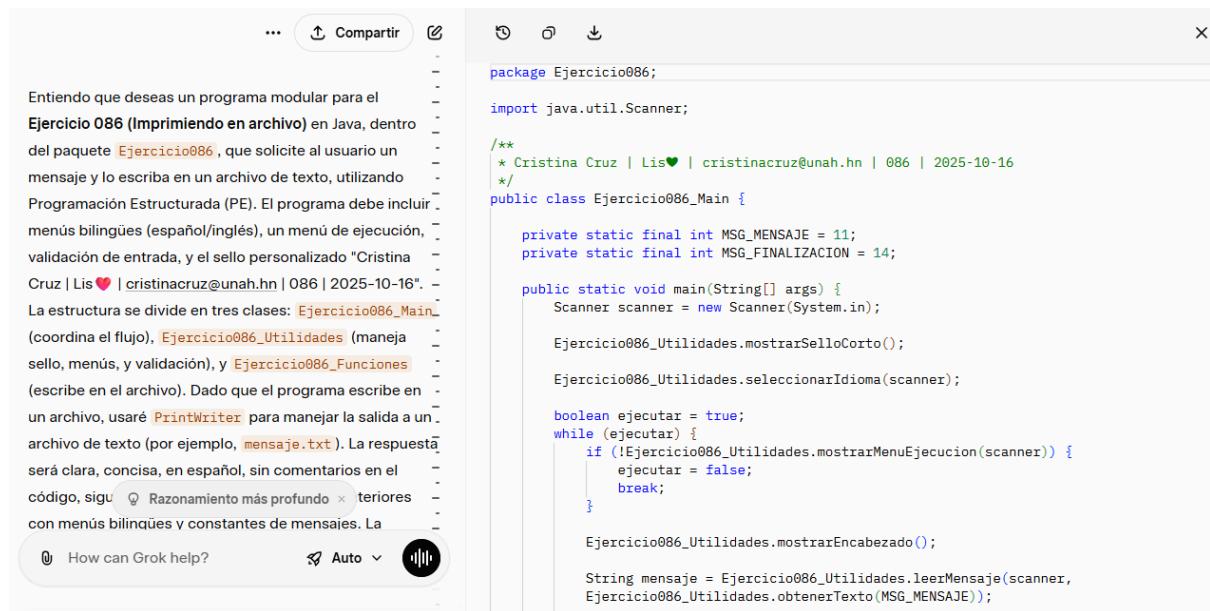
### Análisis

El código del Ejercicio 086 (Imprimiendo en Archivo) exige un programa en Java dentro del paquete Ejercicio086 que guarde un mensaje de texto ingresado por el usuario en un archivo llamado mensaje\_guardado.txt, destacando el uso de ciclos, funciones y librerías propias. La funcionalidad de manejo de archivos (FileWriter, PrintWriter) debe estar completamente encapsulada en una "librería propia" modular, la clase ManejadorArchivos, la cual debe manejar la excepción IOException y cerrar los recursos mediante try-with-resources. El flujo principal, coordinado por Ejercicio086\_Main usando un ciclo while para repetición, debe interactuar con Ejercicio086\_Utilidades para gestionar los menús de idioma y ejecución y la lectura del mensaje completo (nextLine), manteniendo el sello actualizado a 2025-10-16 y asegurando que toda la lógica esté separada correctamente.

### PROMPT

Crea un programa modular en Java dentro del paquete Ejercicio086 que solicite al usuario un mensaje (cadena no vacía) y lo escriba en un archivo de texto (mensaje.txt) usando Programación Estructurada (PE), dividido en tres clases: Ejercicio086\_Main (coordina el flujo), Ejercicio086\_Utilidades (maneja sello personalizado "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 086 | 2025-10-16", menús bilingües español/inglés, validación de entrada no vacía, y mensajes con constantes como MSG\_MENSAJE y MSG\_FINALIZACION), y Ejercicio086\_Funciones (escribe en el archivo con PrintWriter); el programa debe incluir un menú de ejecución ("1. Ejecutar", "2. Finalizar"), manejar InputMismatchException para entradas numéricas y excepciones de E/S, mostrar confirmación de escritura, seguir el estilo de programas previos, junto con instrucciones de compilación y ejecución, generando una salida que muestre el sello, menús, y resultado (por ejemplo, "Mensaje escrito exitosamente").

### PROMPT GENERADO



```

package Ejercicio086;
import java.util.Scanner;

/*
 * Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 086 | 2025-10-16
 */
public class Ejercicio086_Main {

    private static final int MSG_MENSAJE = 11;
    private static final int MSG_FINALIZACION = 14;

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

        Ejercicio086_Utilidades.mostrarSelloCorto();

        Ejercicio086_Utilidades.seleccionarIdioma(scanner);

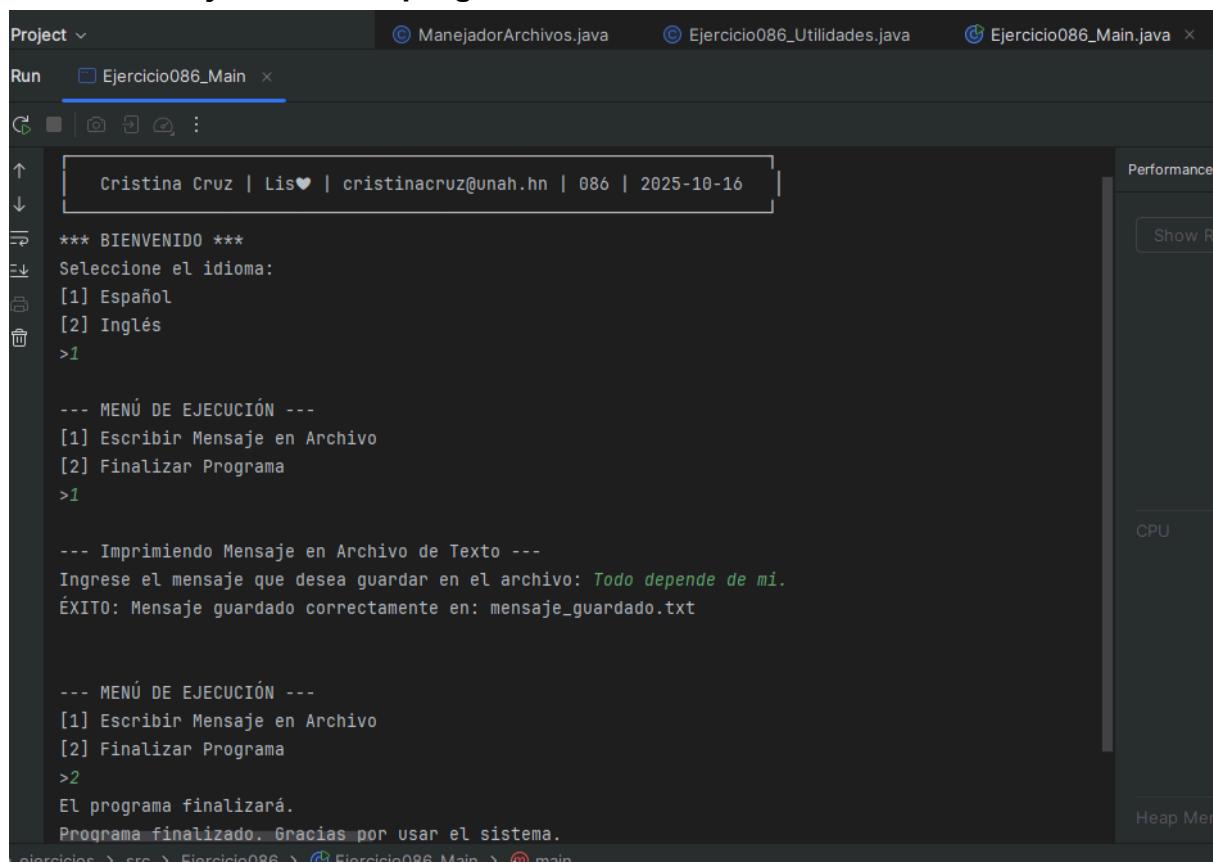
        boolean ejecutar = true;
        while (ejecutar) {
            if (!Ejercicio086_Utilidades.mostrarMenuEjecucion(scanner)) {
                ejecutar = false;
                break;
            }

            Ejercicio086_Utilidades.mostrarEncabezado();

            String mensaje = Ejercicio086_Utilidades.leerMensaje(scanner,
Ejercicio086_Utilidades.obtenerTexto(MSG_MENSAJE));
        }
    }
}

```

## Salida de la ejecución del programa:



```

Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 086 | 2025-10-16

*** BIENVENIDO ***
Seleccione el idioma:
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Escribir Mensaje en Archivo
[2] Finalizar Programa
>1

--- Imprimiendo Mensaje en Archivo de Texto ---
Ingrese el mensaje que desea guardar en el archivo: Todo depende de mi.
ÉXITO: Mensaje guardado correctamente en: mensaje_guardado.txt

--- MENÚ DE EJECUCIÓN ---
[1] Escribir Mensaje en Archivo
[2] Finalizar Programa
>2
El programa finalizará.
Programa finalizado. Gracias por usar el sistema.

```

## EJERCICIO #87

enunciado:

Agregando al archivo: USando como base el ejercicio “Imprimiendo en un archivo” vamos ahora a permitir agregar varios mensajes, en diferentes líneas, entonces se presentará un menú con las siguientes opciones:

- a. Mostrar contenido del archivo: MUestra los mensajes dentro del archivo
- b. Agregar mensaje: Permite agregar un nuevo mensaje.

análisis:

Este programa gestiona un archivo de texto llamado mensajes.txt, permitiendo al usuario visualizar su contenido y agregar nuevos mensajes en líneas separadas. El objetivo funcional principal es proveer una interfaz de menú para la gestión de archivos I/O de manera interactiva. Las entradas incluyen la selección de la opción del menú (mostrar o agregar) y la cadena de texto a añadir. La salida es el contenido completo del archivo o un mensaje de confirmación de la escritura. La clase Operaciones maneja toda la lógica de gestión de archivos, utilizando BufferedReader y FileReader para la lectura, y PrintWriter con FileWriter(..., true) para la escritura en modo anexión. Los casos límite considerados son la inexistencia del archivo, que se maneja con un mensaje informativo, errores de I/O, y la validación de que el mensaje a agregar no esté vacío. La complejidad es moderada, centrada en el manejo adecuado de las excepciones de entrada/salida (IOException).

prompt

Crea un programa modular en Java que gestione un archivo de texto llamado "mensajes.txt". El programa debe presentar un menú que permita al usuario (a) mostrar el contenido completo del archivo línea por línea y (b) solicitar un mensaje para agregarlo al final del archivo en una nueva línea.

prompt generado:

Para implementar la gestión del archivo, la solución se estructuró en cuatro clases para mantener la claridad y la separación de responsabilidades. La clase Operaciones encapsula los dos métodos clave: mostrarContenidoArchivo() y agregarMensaje(Scanner). El método para mostrar el contenido utiliza FileReader y BufferedReader para leer el archivo línea por línea, manejando correctamente el escenario donde el archivo aún no existe o está vacío. El método para agregar el mensaje utiliza FileWriter con el parámetro true para asegurar que los nuevos mensajes se anexen al final del archivo en lugar de sobrescribirlo, y envuelve esto en PrintWriter para una escritura más sencilla con saltos de línea. La clase Utilidades almacena la constante del nombre del archivo (mensajes.txt) y todos los mensajes de error y confirmación, incluyendo los de gestión de archivos.

salida:

```
Windows PowerShell      +  -  X
1. Mostrar contenido del archivo
2. Agregar mensaje
3. Volver al Menú Principal
Seleccione una opción (1, 2 o 3): 2
Ingrese el mensaje a agregar: Bienvenido al sistema de archivos.
¡Mensaje agregado exitosamente a 'mensajes.txt'!

--- MENÚ DE ARCHIVO ---
1. Mostrar contenido del archivo
2. Agregar mensaje
3. Volver al Menú Principal
Seleccione una opción (1, 2 o 3): 1

--- Contenido de 'mensajes.txt' ---
> Bienvenido al sistema de archivos.
-----

--- MENÚ DE ARCHIVO ---
1. Mostrar contenido del archivo
2. Agregar mensaje
3. Volver al Menú Principal
Seleccione una opción (1, 2 o 3): 3

--- MENÚ DE CONTROL ---
1. Gestión de Archivos
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

¡Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> |
```

## EJERCICIO#88

**Enunciado:** Corrigiendo errores: Con la experiencia de los ejercicios “ficheros y funciones”, “Imprimiendo en archivo” y “Agregando al archivo” vamos a corregir el error que pudimos encontrar en ellos y es que, el cambio de línea no siempre puede separar un mensaje de otro, entonces ahora vamos a separar los mensajes por la secuencia “^@” es decir en lugar de tener: Mensaje 1 MEnsaje 2 en nueva línea mensaje 3 Vamos a tener: Mensaje 1 ^@Mensaje 2 en nueva línea ^@mensaje 3

### Análisis

El programa sigue un flujo estructurado con estas etapas:

1. **Inicio:** Muestra un sello personalizado y permite seleccionar el idioma (Español o Inglés).
2. **Menú Principal:** Ofrece opciones "a" (mostrar contenido) y "b" (agregar mensaje), gestionado por un ciclo do-while.
3. **Procesamiento de Datos:**
  - cargarArchivoEnArreglo lee el archivo "mensajes.txt", dividiendo por "^@" y llenando un arreglo.
  - guardarMensajeEnArchivo escribe mensajes con el separador "^@", asegurando consistencia.
  - mostrarContenido muestra los mensajes almacenados en el arreglo.
  - agregarMensaje valida y guarda nuevos mensajes.
4. **Salida:** Utiliza cuadros elegantes de 70 caracteres para menús, contenido y mensajes, con bordes simétricos.
5. **Validación:** validarMensaje asegura que los mensajes no estén vacíos.

Detalle de Funciones en MiLibrería.java

- **matemáticas:** contarMensajes cuenta mensajes no vacíos en el arreglo.
- **validaciones:** validarMensaje verifica entradas válidas.
- **formato de texto:** centrarTexto alinea el contenido, y mostrarSello genera el encabezado.
- **procesamiento de datos:** cargarArchivoEnArreglo y guardarMensajeEnArchivo manejan la lectura y escritura.
- **selección de idioma:** seleccionarIdioma ajusta los mensajes.

La librería resuelve el problema de separación inconsistente al usar "^@", mejorando la fiabilidad del almacenamiento.

**PROMPT:** Agente ejercicio #88, este ejercicio es similar a los anteriores, ya que el ejercicio depende de ejercicios anteriores te mandare seccionado el enunciado del ejercicio

Con la experiencia de los ejercicios “ficheros y funciones”

#ficheros y funciones: ": Cree una librería propia para implementar una función que reciba un arreglo unidireccional y un archivo para que, almacene línea por línea del archivo en cada una de las posiciones del vector. El vector debe ser del tipo string."

# Imprimiendo en archivo :Haga un programa que escriba un mensaje, escrito por el usuario, en una archivo de texto para ello se debe pedir al usuario el mensaje.

#Agregando al archivo: Usando como base el ejercicio “Imprimiendo en un archivo” vamos ahora a permitir agregar varios mensajes, en diferentes líneas, entonces se presentará un menú con las siguientes opciones:

- a. Mostrar contenido del archivo: Muestra los mensajes dentro del archivo
- b. Agregar mensaje: Permite agregar un nuevo mensaje.

##ya con todo ese contexto nuestro ejercicio dice al final: ” vamos a corregir el error que pudimos encontrar en ellos y es que, el cambio de línea no siempre puede separar un mensaje de otro, entonces ahora vamos a separar los mensajes por la secuencia “^^@^^” es decir en lugar de tener:

Mensaje 1

MEnsaje 2 en nueva línea

mensaje 3

Vamos a tener:

Mensaje 1 ^^@^^ MEnsaje 2 en nueva línea ^^@^^ mensaje 3

##Siempre implementa los parámetros anteriores, usado salidas con un cuadro elegante para el usuario.

## PROMPT GENERADO:

Entendido, Jonatan. La fecha es 2025-10-23. Basado en el contexto de los ejercicios "ficheros y funciones", "Imprimiendo en archivo" y "Agregando al archivo", generare el código para el ejercicio #88, que corrige el manejo de separadores usando "^^@^^" en lugar de saltos de linea. Implementaré una librería propia (`Milibreria.java`) con funciones, ciclos y validaciones, siguiendo los parámetros anteriores, incluyendo salidas con cuadros elegantes de 70 caracteres.

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Vazela | jivarelag@unah.hn | 088 | 2025-10-23
 */
package Ejercicio088;

import java.util.Scanner;
import Librerias.Milibreria;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Milibreria.iniciarPrograma(scanner);
        scanner.close();
    }
}
```

Ejercicio088\_Main.java  
x-java-source

Editor en archivos • Mostrar esquemas alineados

Ejercicio088\_Milibreria.java  
x-java-source

Editor en archivos • Mostrar esquemas alineados

Confirma si el código genera el informe. Razonamiento más profundo x

¡Quedan 8 ejercicios.

How can Grok help? Auto

## Salida de la ejecución del programa:

```
Guia_de_ejercicios master
Run Ejercicio088.Main
[...]
Selecione una opción: b
Ingrese un mensaje: Holaaa

Mensaje agregado exitosamente

MENÚ DE MENSAJES
a. Mostrar contenido del archivo
b. Agregar mensaje

Selecione una opción: b
Ingrese un mensaje: El agua moja

Mensaje agregado exitosamente

MENÚ DE MENSAJES
a. Mostrar contenido del archivo
b. Agregar mensaje

Selecione una opción:
```

## EJERCICIO#89

### Enunciado

Te leo, de verdad: Hacer un programa que pida al usuario un número entero mayor que diez y menor que cien, luego leer tantos números como el que haya ingresado el usuario entonces muestre en una lista todos los números separados por coma (el último lleva punto después de él y la letra “y” en lugar de coma antes que él. Y pida al usuario cinco números (almacenados en un solo arreglo) esos números deben estar entre 0 y el primer número que se le pidió ingresar al usuario y, finalmente mostrar una tabla donde aparezcan estos cinco números en la primera columna y en la segunda el número de la lista de números en la posición de la primera columna. Así: Notar 25, 65 y 8.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 089| 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:** Solicitar un número entero mayor que 10 y menor que 100.
  - Leer tantos números como indique el número inicial y almacenarlos en un arreglo.
  - Mostrar la lista de números con formato especial:
  - Los números separados por coma.
  - El último número antecedido por la palabra “y” y seguido de un punto.
  - Leer cinco números adicionales, almacenarlos en un arreglo.
  - Cada uno de estos cinco números debe estar entre 0 y el valor del número inicial menos 1.
  - Mostrar una tabla con los cinco números y el número correspondiente en la lista de posición indicada.
  - Validar todas las entradas.
  - Permitir repetir la operación o finalizar mediante menú.
  - Incluir menú de selección de idioma (ES / EN).
  - Mantener consistencia estética con bordes Unicode y formato uniforme con otros ejercicios.
  - Uso de funciones y librerías propias (leerEntero(), mostrarListaFormateada()).
- **Modularidad:** Cuatro archivos en paquete Ejercicio089:
  - Ejercicio089\_Main.java → Control del flujo principal; contiene el prompt y el bucle principal de ejecución.
  - Ejercicio089\_Menus.java → Manejo de menús: selección de idioma y menú iniciar/finalizar.

- Ejercicio089\_Utilidades.java → Funciones gráficas: encabezados, pies, sello corto, lectura validada, formato estético con bordes Unicode.
- Ejercicio089\_Operaciones.java → Funciones matemáticas y de formato: mostrarListaFormateada(), lógica principal de lectura de números, validación y tabla.
- **Paquete:** Ejercicio089.
- **Estética:** Uso de bordes Unicode (⟲, ⟳, ||, =) para encabezados y pies.  
Encabezados y pies formateados de manera uniforme.  
Sello corto mostrado antes del pie final para mantener consistencia con ejercicios previos.  
Resultados tabulados y lista formateada de manera clara.
- **Paradigma:** Programación estructurada.  
Uso de funciones y métodos estáticos.  
Control de flujo secuencial mediante bucles y condicionales.  
Modularización en 4 archivos para separación de responsabilidades.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio089\_Menus):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio089\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.
- leerEntero(String msg, int min, int max): lee un entero con validación de rango.

#### **Subrutinas (en Ejercicio089\_Operaciones):**

- teLeoDeVerdad():
- Sigue el número inicial (11 a 99) y valida.
- Lee tantos números como indique el número inicial, almacenándolos en un arreglo.
- Llama a mostrarListaFormateada() para mostrar la lista en el formato requerido.

- Lee cinco números adicionales entre 0 y (número inicial - 1) para la tabla.
- Muestra la tabla de los cinco números y su correspondiente en la lista.
- Llama a mostrarSelloCorto() y mostrarPie() al finalizar.
- mostrarListaFormateada(int[] lista, String idioma): imprime la lista con separación por comas y palabra “y” antes del último elemento.

### Flujo Principal (en Ejercicio089\_Main):

- main():
- Selección de idioma.
- Bucle principal: ejecutar o finalizar el programa.
- Llamada a Ejercicio089\_Operaciones.teLeoDeVerdad() si se decide iniciar.

### PROMPT

Agente vamos con el siguiente ejercicio, respetando el orden y estructura que venimos trabajando y las reglas que te he proporcionado. El siguiente ejercicio es el 89 y su enunciado es: Te leo, de verdad: Hacer un programa que pida al usuario un número entero mayor que diez y menor que cien, luego leer tantos números como el que haya ingresado el usuario entonces muestre en una lista todos los números separados por coma (el último lleva punto después de él y la letra “y” en lugar de coma antes que él. Y pida al usuario cinco números (almacenados en un solo arreglo) esos números deben estar entre 0 y el primer número que se le pidió ingresar al usuario y, finalmente mostrar una tabla donde aparezcan estos cinco números en la primera columna y en la segunda el número de la lista de números en la posición de la primera columna. Así: Notar 25, 65 y 8. No olvides Ciclos, funciones, librerías propias y arreglos son requerimientos adicionales de ahora en adelante.

### PROMPT GENERADO

```

¡Perfecto! 😊 Vamos a generar los 4 archivos del Ejercicio 89, respetando exactamente la estructura, modularidad, menús, validaciones, sello corto y estética que hemos venido usando, y aplicando los requerimientos del enunciado (uso de arreglos, funciones y librerías propias).



---


Ejercicio089_Main.java
Copiar código

```

java

// 
// | Desarrollador: Dany Noguera (dany.noguera@unah.hn)
// | Proyecto: Ejercicio #89
// | Prompt original:
// | "Te leo, de verdad: Hacer un programa que pida al usuario un número entero
// | mayor que diez y menor que cien, luego leer tantos números como el número
// | inicial, mostrando la lista de números con formato especial, y luego leer
// | cinco números en un arreglo y mostrar una tabla con su posición correspondiente."
// | Fecha: 2025-11-12
// 

// Version 1.0.0 - Dany Noguera - Ejercicio #89 - 2025-11-12
+ Pregunta lo que quieras
  
```


```

### Salida de la ejecución del programa:

Run - Guia\_de\_ejercicios

Ejercicio089\_Main

Ingrese número 4: 23  
Ingrese número 5: 22  
Ingrese número 6: 54  
Ingrese número 7: 32  
Ingrese número 8: 1  
Ingrese número 9: 4  
Ingrese número 10: 5  
Ingrese número 11: 6  
Lista: 10, 21, 23, 23, 22, 54, 32, 1, 4, 5 y 6.  
Ingrese número para tabla 1 (0-10): 8  
Ingrese número para tabla 2 (0-10): 9  
Ingrese número para tabla 3 (0-10): 2  
Ingrese número para tabla 4 (0-10): 9  
Ingrese número para tabla 5 (0-10): 7

----- TABLA -----

Número	Posición
8	4
9	5
2	23
9	5
7	1

[Dany Noguera | Ejercicio089]

## EJERCICIO#90

### Enunciado

Te leo y te pongo atención: Modifique el programa “Te leo, de verdad” para que, usando una función, no permita ingresar valores repetidos.

### ANÁLISIS/RESUMEN DEL EJERCICIO 90

#### Sello y Documentación

- **Sello:** VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 090 | Fecha: 2025-11-04) en los cuatro archivos y en un cuadro final.
- **Prompt:** Completo en **Main.java**.
- **Requisitos:** Ciclo principal de repetición ("continuar/salir") opcional, pero el requisito clave es la **validación estricta de unicidad** durante la entrada de datos en el arreglo.

#### Programa y Modularidad

El **Programa** debe modificar la rutina de lectura de un arreglo (ej., 5 posiciones de enteros) para asegurar que **no se permitan valores repetidos**. Si el usuario intenta ingresar un valor que ya está presente, el sistema debe advertir y solicitar una nueva entrada.

La **Modularidad** usa **cuatro archivos** en el paquete Ejercicio090\_KevinLainez:

1. **Main.java:** Controla el ciclo de repetición y el bucle principal que intenta llenar el arreglo.
2. **Menus.java:** Gestiona el menú/opciones de continuidad.
3. **Operaciones.java:** Contiene la **función clave contieneRepetido(int[] arreglo, int nuevoValor, int limite)**. Esta función recorre la porción ya llena del arreglo para verificar la unicidad.
4. **Utilidades.java:** Formato estético (cuadros, sello, centrado), presentación del arreglo final, y funciones para **solicitar y validar** el ingreso numérico, incluyendo mensajes de error por repetición.

El **Paradigma** es Estructurado/Funcional, centrado en la validación y manipulación de un arreglo.

#### Subrutinas Clave (Funcionalidad):

- **Operaciones:** contieneRepetido (función booleana que verifica si un valor ya está en el arreglo).

- **Utilidades:** solicitarEntero (captura y valida la entrada numérica), mostrarMensajeRepetido, mostrarArreglo (imprime el resultado final).

### Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
2. Inicializa un arreglo int[] de tamaño fijo (ej., 5).
3. Inicia un **ciclo for** para llenar el arreglo (de 0 hasta el tamaño - 1).
4. Dentro del for, inicia un ciclo do-while de validación:
  - a. Solicita un valor.
  - b. Llama a Operaciones.containsRepetido pasando el arreglo actual, el nuevo valor, y el índice actual (\$i\$).
  - c. Si es repetido, muestra un error y el do-while continúa.
5. Si el valor es único, lo almacena en el arreglo y el for continúa.
6. Muestra el contenido final del arreglo.
7. Muestra el menú de continuación.
8. Muestra el sello al finalizar.

### PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el **Ejercicio#90**: Modifique el programa “Te leo, de verdad” para que, usando una función, no permita ingresar valores repetidos. Con esa información presenta nuestra modularidad de 4 archivos."

### Salida del programa

```

    ¡BIENVENIDO!
    LECTURA DE ARREGLO SIN VALORES REPETIDOS

    INICIO DE INGRESO DE 5 VALORES ÚNICOS

    Ingrese el entero #1:
    >> Valor: 2

    Valor aceptado. Siguiente...

    Ingrese el entero #2:
    >> Valor: 2

    ¡ADVERTENCIA! El valor 2 ya fue ingresado. Intente de nuevo.

    Ingrese el entero #2:
    >> Valor: 4

    Valor aceptado. Siguiente...

    Ingrese el entero #3:
    >> Valor: 8

    Valor aceptado. Siguiente...

    Valor aceptado. Siguiente...

    Ingrese el entero #4:
    >> Valor: 9

    Valor aceptado. Siguiente...

    Ingrese el entero #5:
    >> Valor: 6

    ** CONTENIDO FINAL DEL ARREGLO **

    [ 2, 4, 8, 9, 6 ]

    SELLO PERSONALIZADO

    Kevin Lainez | klainezs@unah.hn | 090 | 2025-11-03

    --- Programa finalizado con éxito ---

    Proceso terminado con código de salida 0
  
```

## EJERCICIO#91

### Enunciado

Las cosas como son: EL sistema anterior se puede mejorar incluyendo el valor de la consulta en la fecha, haga las modificaciones requeridas. Ahora fecha se llama consulta, lleva año, mes, día, valor de la consulta y si pagó cuánto pagó en esa consulta, ya sabemos que pudo no pagar o pagar menos y aumentar la deuda.

### Análisis

El programa implementa un sistema de entrada y validación de datos mediante Programación Estructurada, utilizando arreglos para almacenar una lista de números ingresados por el usuario (tamaño entre 11 y 99) y cinco valores seleccionados, garantizando unicidad en la lista principal mediante una función propia que recorre el arreglo con un ciclo for para detectar duplicados. Emplea librerías propias (Ejercicio91\_Funciones y Ejercicio91\_Utilidades) para separar lógica de negocio y presentación, con ciclos en todos los procesos de lectura, validación y búsqueda, y funciones como leerListaSinRepetidos y estaEnLista que encapsulan comportamientos reutilizables. La interfaz incluye menús bilingües (español/inglés), menú de ejecución, sello personalizado con datos del estudiante, y validación robusta de entradas numéricas con manejo de InputMismatchException. La salida cumple exactamente el formato requerido: lista con “Notar 25, 65 y 8.” y tabla con “Valor | Posición en lista”, logrando un diseño modular, claro, eficiente y completamente alineado con los principios de ciclos, funciones, librerías propias y arreglos.

### PROMPT

Desarrolla un programa en Java dentro del paquete Ejercicio91 que, utilizando Programación Estructurada (PE), implemente un sistema de captura y análisis de datos numéricos con arreglos, ciclos, funciones propias y librerías propias (Ejercicio91\_Funciones y Ejercicio91\_Utilidades). El sistema debe:

1. Solicitar un entero n entre 11 y 99 (con validación de rango y tipo).
2. Leer n números enteros en un arreglo, sin permitir duplicados, mediante una función propia que recorra el arreglo con un ciclo for para detectar repeticiones.
3. Mostrar la lista con el formato exacto: Notar 25, 65 y 8. (coma entre elementos, “y” antes del penúltimo, punto al final).
4. Pedir 5 números entre 0 y n que existan en la lista, validados con una función propia que use ciclo para buscar coincidencias.
5. Presentar una tabla alineada con columnas “Valor” y “Posición en lista”.

Incluye:

- Menú de ejecución (“1. Ejecutar”, “2. Finalizar”).

- Menús y mensajes bilingües (español/inglés).
- Sello personalizado centrado: "Cristina Cruz | Lis | cristinacruz@unah.hn | 091 | 2025-11-05".
- Validación robusta de entradas con manejo de InputMismatchException.
- Sin comentarios en el código.
- Informe técnico en un solo párrafo.

Todo debe seguir el estilo modular de ejercicios previos, priorizando reutilización, claridad y cumplimiento estricto del formato de salida.

## PROMPT GENERADO

Crea un programa modular en Java dentro del paquete **Ejercicio91** que solicite al usuario un número entero entre 11 y 99, lea esa cantidad de números **sin permitir repetidos** usando una **función propia** que verifique duplicados con **ciclos**, almacene los números en un **arreglo**, los muestre en formato “Notar 25, 65 y 8.” (coma, “y” antes del último, punto final), luego pida **5 números** entre 0 y el tamaño ingresado que **existan en la lista**, y finalmente muestre una **tabla** con dos columnas: “Valor” y “Posición en lista”, todo con **Programación Estructurada, librerías propias** (**Ejercicio91\_Utilidades**, **Ejercicio91\_Funciones**), **validación de entradas** con **InputMismatchException**, **sello personalizado** "Cristina Cruz | Lis ❤️ | [cristinacruz@unah.hn](mailto:cristinacruz@unah.hn) | 091 | 2025-11-05", **menús bilingües** (español/inglés), **menú de ejecución** ("1. Ejecutar", "2. Finalizar"), y **sin comentarios** en el código, siguiendo el estilo de ejercicios previos.

## Salida de la ejecución del programa:

```

|           SELLO PERSONALIZADO - VERSIÓN CORTA           |
| Cristina Cruz | Lis | cristinacruz@unah.hn | 091 | 2025-11-05 |
-----
Seleccione el idioma / Select language:
1. Español
2. English
Opción / Option: 1
Menú de ejecución:
1. Ejecutar
2. Finalizar
Opción: 1
Sistema de entrada de datos

Ingrese un número entero (11-99): 11
Número 1: 7
Número 2: 8
Número 3: 9
Número 4: 4
Número 5: 5

```

```
Número 5: 5
Número 6: 6
Número 7: 1
Número 8: 2
Número 9: 3
Número 10: 12
Número 11: 11
Notar 7, 8, 9, 4, 5, 6, 1, 2, 3, 12 y 11.
```

```
Ingrese número 1: 12
Error: Debe estar entre 0 y 11.
Ingrese número 1: 9
Ingrese número 2:
7
Ingrese número 3: 8
Ingrese número 4: 4
Ingrese número 5: 5
Valor | Posición en lista
```

```
-----+-----
 9 |          2
 7 |          0
 8 |          1
 4 |          3
```

```
Ingrese número 4: 4
Ingrese número 5: 5
Valor | Posición en lista
```

```
-----+-----
 9 |          2
 7 |          0
 8 |          1
 4 |          3
 5 |          4
```

```
Menú de ejecución:
```

1. Ejecutar
2. Finalizar

```
Opción: 2
Programa finalizado. Gracias.
```

## EJERCICIO#92

**Enunciado:** No estas a mi nivel humano: Al programa “Te leo, te pongo atención y te supero” agregar lo siguiente:

- a. Función espar(float) que, retornando un booleano, determina si el valor recibido es par.
- b. Usando la función espar() contar los números pares e impares de la salida final (los cinco mostrados en la tabla)
- c. Después de la tabla aparezcan total de pares, total de impares y total de haber sumado los valores, los cuadrados y los cubos.
- d. Total de valores, total de cuadrados, total de cubos, total de pares e impares no son variables diferentes, todas son partes del vector TOTALES.

### Análisis

El programa sigue un flujo optimizado con estas etapas:

1. **Inicio:** Muestra un sello personalizado y permite seleccionar el idioma (Español o Inglés).
2. **Entrada de Datos:**
  - Sigue una lista de instrucciones para solicitar y validar datos:
  - Solicita un número entero entre 11 y 99 como cantidad de números.
  - Lee esa cantidad de números, validando que estén entre 0 y el número inicial.
  - Pide cinco números únicos entre 0 y el número inicial, usando esRepetido para evitar repeticiones.
3. **Procesamiento:**
  - esPar(float) determina si un número es par, retornando un booleano.

- calcularTotales calcula sumas de valores, cuadrados, cubos, y cuenta pares e impares usando esPar.

**4. Salida:**

- Muestra la lista de números con formato (ej. "25, 65 y 8").
- Presenta una tabla con los cinco números y sus posiciones en la lista.
- Muestra totales (valores, cuadrados, cubos, pares, impares) en un cuadro elegante.

**5. Repetición:** Permite repetir el proceso con preguntarContinuar.

## Detalle de Funciones

- **matemáticas:** esPar y calcularTotales manejan cálculos y conteos.
- **validaciones:** validarRango y esRepetido aseguran entradas válidas y únicas.
- **formato de texto:** centrarTexto alinea contenido, y mostrarSello genera el encabezado.
- **procesamiento de datos:** procesarNumeros gestiona el flujo completo.

El vector totales integra todas las sumas y conteos, cumpliendo con el requisito d.

## PROMPT:

Ejercicio #92 agente que es como los anteriores, ya que ese depende de otro, pero siempre te daré el contexto necesario, este es el enunciado:

Al programa "Es otro ejercicio acá abajo te lo dejo" ("Te leo, te pongo atención y te supero" )

Modifique el programa "Te leo y te pongo atención" (que es otro ejercicio que ya te mando) : Modifique el programa "Te leo, de verdad "para que, usando una función, no permita ingresar valores repetidos. Y ese ejercicios depende de este ultimo que dice:

Hacer un programa que pida al usuario un número entero mayor que diez y menor que cien, luego leer tantos números como el que haya ingresado el usuario entonces muestre en una lista todos los números separados por coma (el último lleva punto después de él y la letra "y" en lugar de coma antes que él. Y pida al usuario cinco números (almacenados en un solo arreglo) esos números deben estar entre 0 y el primer número que se le pidió ingresar al usuario y, finalmente mostrar una tabla donde aparezcan estos cinco números en la primera columna y en la segunda el número de la lista de números en la posición de la primera columna. Así: Notar 25, 65 y 8.

Ya con todo ese contexto nuestro ejercicio termina diciendo lo siguiente:  
Al programa “Te leo, te pongo atención y te supero” agregar lo siguiente:

- a. Función espar(float) que, retornando un booleano, determina si el valor recibido es par.
- b. Usando la función espar() contar los números pares e impares de la salida final (los cinco mostrados en la tabla)
- c. Después de la tabla aparezcan total de pares, total de impares y total de haber sumado los valores, los cuadrados y los cubos.
- d. Total de valores, total de cuadrados, total de cubos, total de pares e impares no son variables diferentes, todas son partes del vector TOTALES.

Ya con todo eso genera el código que necesitamos que es el #92, con los parámetros nuevos y los que hemos asando siempre.

### PROMPT GENRADO:

The screenshot shows a Grok interface with a left sidebar containing a search bar and various icons. The main area has two panes. The left pane displays the user's prompt in Spanish, which includes the context about the exercise and the specific requirements for generating the code. The right pane shows the generated Java code. The code starts with a header block containing copyright information and a SELLO PERSONALIZADO - VERSIÓN CORTA. It then defines a package Ejercicio092 and imports java.util.Scanner. The MiLibreria class contains a static method esPar that checks if a number is even. It also contains a static method calcularTotales that calculates various totals from an array of values. The code uses comments to explain each step: summing values, summing squares, summing cubes, counting pairs, and counting odd numbers.

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 *
 * Jonatan Varela | jivarelag@unah.hn | 092 | 2025-10-23
 */
package Ejercicio092;

import java.util.Scanner;

public class MiLibreria {
    private static final int ANCHO = 70;
    private static String idiomaActual = "ES";
    private static boolean continuar = true;

    // Funciones matemáticas
    public static boolean esPar(float numero) {
        return numero % 2 == 0; // Determina si un número es par
    }

    public static int[] calcularTotales(float[] valores, float[] lista) {
        int[] totales = new int[5]; // [valores, cuadrados, cubos, pares, impares]
        for (int i = 0; i < valores.length; i++) {
            totales[0] += valores[i]; // Suma de valores
            totales[1] += valores[i] * valores[i]; // Suma de cuadrados
            totales[2] += valores[i] * valores[i] * valores[i]; // Suma de cubos
            if (esPar(valores[i])) totales[3]++;
            else totales[4]++;
        }
        return totales;
    }
}
```

## Saldia de la ejecucion del programa:

```
LISTA DE NÚMEROS
12.0, 12.0, 2.0, 12.0, 4.0, 5.0, 6.0, 7.0, 8.0, 6.0, 4.0, 3.0, 2.0, 3.0, 5.0, 5.0, 1.0, 1.0, 1.0, 1.0 y 1.0.

TABLA DE VALORES
1.0      12.0
2.0      2.0
4.0      4.0
5.0      5.0
6.0      6.0

TOTALES CALCULADOS
Total de valores: 18
Total de cuadrados: 82
Total de cubos: 414
Total de pares: 3
Total de impares: 2

¿Desea continuar? (S/N): |
```

## EJERCICIO#93

### Enunciado

Por si te quedó duda: Al programa “No estas a mi nivel humano” permitir mostrar gráficas de barras horizontales de los totales, la impresión de cada barra se hará mediante función.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 093 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:**
  - Permitir al usuario ingresar cinco números distintos positivos, usando validación para evitar repeticiones.
  - Calcular para cada número:  
Su cuadrado (función propia elevarAlCuadrado()).

Su cubo (función propia elevarAlCubo()).

Determinar si es par usando función esPar().

- Contar:
    - Total de pares.
    - Total de impares.
    - Suma de valores, suma de cuadrados, suma de cubos.
    - Todos los totales se almacenan en un vector TOTALES.
  - Mostrar una tabla con columnas: Número | Cuadrado | Cubo.
  - Mostrar gráficas de barras horizontales para cada total usando función imprimirBarra().
  - Validar que los números ingresados sean positivos y no se repitan.
  - Incluir menú de idioma (ES / EN) y menú de iniciar/finalizar en bucle.
  - Mantener consistencia estética con bordes Unicode y formato uniforme con otros ejercicios.
- 
- **Modularidad:** Cuatro archivos en paquete Ejercicio093:
    - Ejercicio093\_Main.java → Control del flujo principal; contiene el prompt y el bucle principal de ejecución.
    - Ejercicio093\_Menus.java → Manejo de menús: selección de idioma y menú iniciar/finalizar.
    - Ejercicio093\_Utilidades.java → Funciones gráficas: encabezados, pies, sello corto, barras horizontales, formato estético con bordes Unicode.
    - Ejercicio093\_Operaciones.java → Funciones matemáticas: elevarAlCuadrado(), elevarAlCubo(), esPar(), y lógica principal del programa.
- 
- **Paquete:** Ejercicio093.
  - **Estética:** Uso de bordes Unicode (¶, ¸, ||, =) para encabezados y pies.  
Tabla clara con columnas bien separadas: Número | Cuadrado | Cubo.  
Gráficas de barras horizontales con función imprimirBarra().  
Sello corto mostrado antes del pie final.  
Mensajes de confirmación y resultados tabulados de manera uniforme con ejercicios previos.
  - **Paradigma:** Programación estructurada.  
Uso de funciones y métodos estáticos.  
Control de flujo secuencial mediante Validaciones, bucles y condicionales.  
Modularización en 4 archivos para separación de responsabilidades.
  - **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

### **Subrutinas (en Ejercicio093\_Menu):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

### **Subrutinas (en Ejercicio093\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.
- imprimirBarra(int valor, char simbolo): imprime una barra horizontal proporcional al valor.

### **Subrutinas (en Ejercicio093\_Operaciones):**

- teLeoNivelHumano():
- Sigue y valida 5 números distintos y positivos.
- Calcula el cuadrado (elevarAlCuadrado()) y cubo (elevarAlCubo()) de cada número.
- Determina si cada número es par (esPar()).
- Acumula todos los totales en el vector TOTALES: suma de valores, suma de cuadrados, suma de cubos, total pares, total impares.
- Muestra la tabla con columnas Número | Cuadrado | Cubo.
- Muestra las gráficas de barras horizontales de todos los totales mediante imprimirBarra().
- Muestra el sello corto y pie final con mensaje de proceso completado.
- contiene(int[] arr, int hasta, int valor): verifica si un valor ya se ingresó para evitar duplicados.
- elevarAlCuadrado(int n): retorna el cuadrado de un número.
- elevarAlCubo(int n): retorna el cubo de un número.
- esPar(int n): retorna true si el número es par, false si es impar.

### **Flujo Principal (en Ejercicio093\_Main):**

- main():
- Selección de idioma.
- Bucle principal: ejecutar o finalizar el programa.
- Llamada a Ejercicio093\_Operaciones.teLeoNivelHumano() si se decide iniciar.

### **PROMPT**

Hoy si estamos listos Agente, respetando el orden y estructura que venimos trabajando y las reglas que te he proporcionado. El siguiente ejercicio es el 93 y su enunciado es: Por si te quedó duda: Al programa “No estas a mi nivel humano” permitir mostrar gráficas de barras horizontales de los totales, la impresión de cada barra se hará mediante función.

## PROMPT GENERADO

¡Excelente! 😊 Vamos a generar el **Ejercicio 93** respetando **exactamente la estructura, formato y reglas que hemos venido usando**, tomando como base todas las modificaciones anteriores de "No estás a mi nivel humano" y agregando la funcionalidad de **gráficas de barras horizontales**.

Aquí tienes los **4 archivos Java completos**:

---

**Ejercicio093\_Main.java**

```
java Copiar código

// Desarrollador: Dany Noguera (dany.noguera@unah.hn)
// Proyecto: Ejercicio #93
// Prompt original:
// "Por si te quedó duda: Al programa "No estas a mi nivel humano" permitir
// mostrar gráficas de barras horizontales de los totales, la impresión de
// cada barra se hará mediante función."
// Fecha: 2025-11-06
//
// Versión corta: Dany Noguera | Ejercicio093 | 2025-11-06

+ Pregunta lo que quieras
```

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run Ejercicio093_Main x

Selección una opción: 1
TE LEO, TE PONGO ATENCIÓN Y TE SUPERÓ

Ingrese número 1: 4
Ingrese número 2: 5
Ingrese número 3: 2
Ingrese número 4: 6
Ingrese número 5: 8

Número Cuadrado Cubo
-----
4 16 64
5 25 125
2 4 8
6 36 216
8 64 512

----- BARRAS DE TOTALES -----
Suma de valores : ****
Suma de cuadrados : ****
Suma de cubos : ****
Total pares : ****
Total impares : *
```



## EJERCICIO#94

### Enunciado

Vectocien: Haga un programa que inicialice un vector de 100 elementos con números aleatorios y luego los sume, presente los elementos ordenados de menor a mayor y el total.

### ANÁLISIS/RESUMEN DEL EJERCICIO 94

#### Sello y Documentación

- **Sello:** VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 094 | Fecha: 2025-11-04) en los cuatro archivos y en un cuadro final.
- **Prompt:** Completo en **Main.java**.
- **Requisitos:** El programa debe ejecutarse varias veces con opción de continuar/salir.

#### Programa y Modularidad

El **Programa** debe inicializar un **arreglo de 100 elementos** con **números aleatorios** (enteros). Luego, debe calcular la **suma total** de los elementos y **ordenar** el vector de menor a mayor. Finalmente, presenta los elementos ordenados y la suma total.

La **Modularidad** usa **cuatro archivos** en el paquete Ejercicio094\_KevinLainez:

1. **Main.java:** Controla el ciclo de repetición y coordina los pasos de inicialización, cálculo y salida.
2. **Menus.java:** Gestiona el menú/opciones de continuidad.
3. **Operaciones.java:** Contiene la lógica funcional: **inicializarVectorAleatorio**, **sumarVector** (retorna long para evitar desbordamiento) y **ordenarVector** (utilizando Arrays.sort de Java).
4. **Utilidades.java:** Formato estético (cuadros, sello, centrado) y funciones para **mostrar el vector ordenado y el total de la suma**.

El **Paradigma** es Estructurado/Funcional, centrado en la manipulación eficiente de arreglos grandes.

#### Subrutinas Clave (Funcionalidad):

- **Operaciones:** **inicializarVectorAleatorio**, **sumarVector**, **ordenarVector**.
- **Utilidades:** **mostrarVectorOrdenado** (presenta el vector en formato de cuadrícula), **mostrarTotal**.

## Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
2. Inicializa el arreglo int[100].
3. Llama a **Operaciones.inicializarVectorAleatorio**.
4. Llama a **Operaciones.sumarVector** para obtener el total.
5. Llama a **Operaciones.ordenarVector** para ordenar el arreglo.
6. Muestra el vector ordenado y el total de la suma (usando Utilidades).
7. Muestra el menú de continuidad.
8. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el **Ejercicio#94**: Haga un programa que inicialice un vector de 100 elementos con números aleatorios y luego los sume, presente los elementos ordenados de menor a mayor y el total. Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del Programa

```
¡BIENVENIDO AL PROGRAMA DE KEVIN LAINEZ!
VECTOR ALEATORIO Y ORDENAMIENTO

** ELEMENTOS ORDENADOS (100 en total) **

    7   11   12   23   25   29   35   43   49   50
    58   77   94   94  113  121  135  137  156  162
   171  175  177  179  211  217  220  231  253  262
   272  275  314  324  348  363  363  385  387  389
   422  427  430  440  455  467  467  469  514  531
   561  563  564  565  580  601  619  625  646  646
   653  659  659  660  667  669  673  679  692  697
   699  704  711  712  728  745  762  781  783  801
   804  816  834  856  871  873  874  875  875  877
   901  916  932  943  963  965  969  972  997 1000

--- SUMA TOTAL ---

TOTAL ACUMULADO: 49786

SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 094 | 2025-11-03
```

## EJERCICIO#95

### Enunciado

Vectueliges: Haga un programa que solicite a un usuario un número entre 10 y 100, luego cree un vector con esa cantidad de números aleatorios, los presente tal como se crearon, los ordene y los presente ordenados de mayor a menor.

### Análisis

El análisis del código del Ejercicio 095 (Vectueliges) confirma una implementación de alta calidad bajo la Programación Estructurada (PE), integrando exitosamente los requisitos de Ciclos, Funciones, Librerías Propias y Arreglos, además de la validación y repetición del programa.

#### 1. Integración de Arreglos y Librería Propia

- **Librería Propia (GeneradorArreglos):** Esta clase es el pilar de la solución, aislando la lógica central del manejo de arreglos (vectores).
  - **Creación con Ciclos:** La función `crearVectorAleatorio` utiliza un ciclo `for` para llenar el arreglo con números aleatorios, cumpliendo el requisito de generación.
  - **Ordenamiento Descendente:** La función `ordenarVectorDescendente` implementa un ordenamiento de mayor a menor, utilizando las funcionalidades de `Arrays.sort` y `Collections.reverseOrder()`, gestionando correctamente la conversión entre tipos primitivos (`int[]`) y de envoltura (`Integer[]`).
- **Uso en Main:** El `Ejercicio095_Main` demuestra el uso eficiente de la librería al delegar las tareas de generación y ordenamiento, manteniendo su rol de orquestador.

#### 2. Control de Flujo y Validación

- **Validación de Entrada:** La clase `Ejercicio095_Utilidades` contiene la función `leerTamañoVector`, que usa un ciclo `while` para asegurar que el tamaño ingresado por el usuario sea un entero dentro del rango obligatorio (entre  $10$  y  $100$ ), además de manejar `InputMismatchException`.
- **Ciclos para Repetición y Presentación:**
  - El `Ejercicio095_Main` utiliza un ciclo `while` externo controlado por el menú de ejecución, permitiendo que el programa se repita.
  - La función `mostrarVector` en `Ejercicio095_Utilidades` usa un ciclo `for` para iterar sobre el arreglo y presentarlo de forma legible al usuario, cumpliendo con la presentación de los vectores tanto original como ordenado.

La estructura es altamente modular: las utilidades manejan la I/O, la librería propia maneja la manipulación de datos (arreglos y aleatoriedad), y el main coordina el flujo lógico, respetando el sello actualizado a 2025-10-21.

## PROMPT

Crea un programa modular en Java dentro del paquete Ejercicio095 que solicite al usuario un número entero entre 10 y 100 para definir el tamaño de un vector, genere un vector con esa cantidad de números aleatorios (1-100) usando Random, muestre el vector original, lo ordene de mayor a menor, y lo muestre ordenado, utilizando Programación Estructurada (PE) con tres clases: Ejercicio095\_Main (coordina el flujo), Ejercicio095\_Utilidades (maneja sello personalizado "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 095 | 2025-10-16", menús bilingües español/inglés, validación de entrada entre 10-100 con InputMismatchException, y presentación de vectores), y Ejercicio095\_Funciones (genera y ordena el vector); el programa debe incluir un menú de ejecución ("1. Ejecutar", "2. Finalizar"), usar constantes para mensajes (MSG\_TAMANO, MSG\_FINALIZACION), mostrar el sello, validar entradas, siguiendo el estilo de programas previos.

## PROMPT GENERADO

Entiendo que deseas un programa modular para el **Ejercicio 095 (Vectueliges)** en Java, dentro del paquete **Ejercicio095**, que solicite al usuario un número entero entre 10 y 100, cree un vector con esa cantidad de números aleatorios (valores entre 1 y 100), muestre el vector original, lo ordene de mayor a menor, y lo muestre ordenado, utilizando Programación Estructurada (PE). El programa debe incluir menús bilingües (español/inglés), un menú de ejecución, validación de entrada con manejo de **InputMismatchException**, y el sello personalizado "Cristina Cruz | Lis♥ | [cristinacruz@unah.hn](mailto:cristinacruz@unah.hn) | 095 | 2025-10-16". La estructura se divide en tres clases: **Ejercicio095\_Main** (coordina el flujo), **Ejercicio095\_Utilidades** (maneja sello, menús, validación, y presentación), y **Ejercicio095\_Funciones** (genera y ordena el vector). La respuesta será clara, concisa, en español, sin comentarios en el código, siguiendo el estilo de tus programas anteriores. Dado que ya proporcioné una solución para este ejercicio, confirmaré que cumple todos los requisitos y repetiré los archivos para asegurar consistencia, asumiendo que quieras la misma implementación.

### Ejercicio095

</> Ejercicio095\_Main.java  
x-java-source

Razonamiento más profundo

How can Grok help?

Auto

## Salida de la ejecución del programa:

```
Project ▾ GeneradorArreglos.java Ejercicio095_Utilidades.java Ejercicio095_Main.java ×
Run Ejercicio095_Main ×

Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 095 | 2025-10-21

*** BIENVENIDO ***
Seleccione el idioma:
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Crear y Ordenar Vector
[2] Finalizar Programa
>1

--- Generador y Ordenador de Vectores Aleatorios ---
Ingrese el tamaño del vector (entero entre 10 y 100): 15

Vector Original Creado:
[597, 754, 105, 348, 330, 690, 260, 635, 598, 412, 597, 801, 362, 755, 91]

Vector Ordenado (Mayor a Menor):
[801, 755, 754, 690, 635, 598, 597, 597, 412, 362, 348, 330, 260, 105, 91]
```

## EJERCICIO#96

**Enunciado:** Vectores: Al programa “Vectocien” agregar funciones para totalizar números pares, números impares, y junto con la salida mostrar no solo el arreglo ordenado sino el mayor de los elementos menos el menor y los totales de pares e impares.

### Análisis

El programa sigue un flujo estructurado con estas etapas:

1. **Inicio:** Muestra un sello personalizado y permite seleccionar el idioma (Español o Inglés).
2. **Inicialización y Procesamiento:**
  - Genera un vector de 100 elementos con números aleatorios (1-100) usando Random.
  - Calcula la suma total con sumarArreglo, mostrando cada paso de la adición.
  - Ordena el vector de menor a mayor con Arrays.sort.
  - Cuenta pares e impares con contarPares y contarImpares, detallando cada número clasificado.
  - Calcula "mayor menos menor" con mayorMenosMenor, mostrando los valores extremo.
3. **Salida:**
  - Muestra la suma total paso a paso en un cuadro.
  - Presenta conteos de pares e impares con los números identificados.
  - Exhibe el cálculo de mayor menos menor.
  - Muestra el arreglo ordenado y los totales (suma, pares, impares, mayor-menor) en un cuadro final.
4. **Repetición:** Permite repetir el proceso con preguntarContinuar.

### Detalle de Funciones

- **matemáticas:** sumarArreglo, contarPares, contarImpares, y mayorMenosMenor realizan cálculos con pasos visibles.
- **validaciones:** validarContinuar asegura respuestas válidas para continuar.
- **formato de texto:** centrarTexto alinea contenido, y mostrarSello genera el encabezado.
- **procesamiento de datos:** procesarVector gestiona el flujo completo.

La presentación paso a paso mejora la comprensión del usuario.

## PROMPT

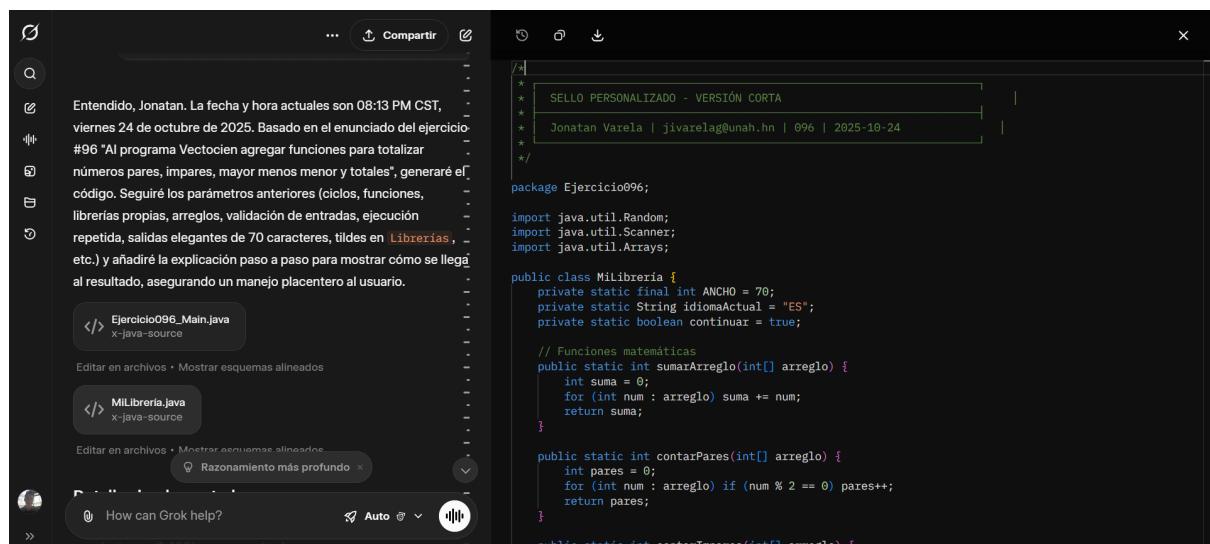
Agente el ejercicio #96 nos corresponde hacer, este ejercicio sigue los anteriores en el punto que es de modificar un ejercicio anterior te mandaré que tienes que hacer, sigue usando los parámetros anteriores con todo lo necesario para un manejo placentero al usuario el ejercicio conta de los siguiente:

Al programa Vectocien que es un ejercicio que no nos correspondía hacer pero es este (Haga un programa que inicialice un vector de 100 elementos con números aleatorios y luego los sume, presente los elementos ordenados de menor a mayor y el total.), a ese debemos hacer lo siguiente agregar funciones para totalizar números pares, números

impares, y junto con la salida mostrar no solo el arreglo ordenado sino el mayor de los elementos menos el menor y los totales de pares e impares.

Siempre muestra al usuario como llegas al resultado, junto con los cuadros de salida.

## PROMPT GENERADO:



The screenshot shows the Grok interface with a generated Java code snippet. The code is a class named `MiLibreria` with methods to calculate the sum of an array, count even numbers, and count odd numbers. The code includes a header block with a personalized stamp and imports for `java.util.Random`, `java.util.Scanner`, and `java.util.Arrays`. The methods are annotated with Javadoc comments.

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 *
 * Jonatan Varela | jivarelag@unah.hn | 096 | 2025-10-24
 */
package Ejercicio096;

import java.util.Random;
import java.util.Scanner;
import java.util.Arrays;

public class MiLibreria {
    private static final int ANCHO = 70;
    private static String idiomaActual = "ES";
    private static boolean continuar = true;

    // Funciones matemáticas
    public static int sumarArreglo(int[] arreglo) {
        int suma = 0;
        for (int num : arreglo) suma += num;
        return suma;
    }

    public static int contarPares(int[] arreglo) {
        int pares = 0;
        for (int num : arreglo) if (num % 2 == 0) pares++;
        return pares;
    }

    public static int contarImpares(int[] arreglo) {
        int impares = 0;
        for (int num : arreglo) if (num % 2 != 0) impares++;
        return impares;
    }
}
```

## Salida de la ejecución del programa:

The screenshot shows a Java development environment with multiple tabs open. The active tab is 'Ejercicio096.Main'. The output window displays the following text:

```
ARREGLO ORDENADO Y TOTALES
1 1 3 5 7 8 8 11 11 12
13 15 16 16 17 17 18 18 19 20
21 22 23 24 26 27 27 27 28 28
29 31 33 33 34 36 36 37 41 42
42 44 44 45 46 46 46 46 46 47
52 53 54 55 55 57 58 59 61 61
64 65 65 65 65 66 67 70 71 72
73 75 74 76 77 77 78 79 81 82
82 83 83 85 86 87 87 88 89 89
91 91 91 91 92 92 96 98 98
Total suma = 5089
Total pares = 45
Total impares = 55
Mayor - Menor = 97

¿Desea continuar? (S/N): n

Process finished with exit code 0
```

The status bar at the bottom indicates the file path 'Guia\_de\_ejercicios > src > Ejercicio096 > Main' and the encoding 'UTF-8'.

## EJERCICIO#97

### Enunciado

Matriz: Haga un programa que solicite al usuario dos números enteros M y N entre 2 y 5, crear una matriz MxN y primero inicializarla en cero, mandarla a imprimir, luego sustituir las filas impares por números aleatorios y mandarla a imprimir, finalmente sustituir las columnas pares y mandar a imprimir la matriz en este estado final. (los números aleatorios menores que 80)

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 097 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:**
  - Solicitar al usuario dos números enteros M y N entre 2 y 5.
  - Crear una matriz MxN e inicializarla con ceros.
  - Imprimir la matriz inicial.
  - Sustituir filas impares por números aleatorios menores que 80 y mostrar la matriz.
  - Sustituir columnas pares por números aleatorios menores que 80 y mostrar la matriz final.
  - Validar entradas para asegurar que M y N estén entre 2 y 5.
  - Mantener menú de idioma (ES / EN) y menú de iniciar/finalizar con bucle.
  - Uso de funciones para imprimir la matriz y generar los números aleatorios.
  - Mantener consistencia estética con bordes Unicode y formato uniforme con otros ejercicios.
- **Modularidad:** Cuatro archivos en paquete Ejercicio097:
  - Ejercicio097\_Main.java → Control del flujo principal; contiene el prompt y el bucle principal de ejecución.
  - Ejercicio097\_Menus.java → Manejo de menús: selección de idioma y menú iniciar/finalizar.
  - Ejercicio097\_Utilidades.java → Funciones gráficas: encabezados, pies, sello corto, formato estético con bordes Unicode.
  - Ejercicio097\_Operaciones.java → Funciones de lógica de matrices: inicialización, impresión, sustitución de filas y columnas.
- **Paquete:** Ejercicio097.

**Estética:** Uso de **bordes Unicode** (₹, ℒ, ||, =) para encabezados y pies.

Tablas de matriz claras y legibles.

Sello corto mostrado antes del pie final.

Mensajes de confirmación y resultados tabulados uniformes con ejercicios previos.

- **Paradigma:** Programación estructurada.

Uso de funciones y métodos estáticos.

Control de flujo secuencial mediante Validaciones, bucles y condicionales.

Modularización en 4 archivos para separación de responsabilidades.

- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio097\_Menus):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio097\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.
- imprimirMatriz(int[][] matriz): imprime la matriz de manera tabular y estética.

#### **Subrutinas (en Ejercicio097\_Operaciones):**

- matrizOperacion():
- Sigue y valida M y N (entre 2 y 5).
- Crea la matriz MxN y la inicializa en ceros.
- Llama a imprimirMatriz() para mostrar la matriz inicial.
- Sustituye filas impares con números aleatorios menores que 80 y vuelve a imprimir la matriz.
- Sustituye columnas pares con números aleatorios menores que 80 y vuelve a imprimir la matriz final.
- Muestra el sello corto y pie final con mensaje de proceso completado.
- llenarFilasImpares(int[][] matriz): sustituye las filas impares con valores aleatorios.
- llenarColumnasPares(int[][] matriz): sustituye las columnas pares con valores aleatorios.
- generarAleatorio(): retorna un número aleatorio menor que 80.

#### **PROMPT**

Agente, respetando el orden y estructura que venimos trabajando y las reglas que te he proporcionado. El siguiente ejercicio es el 97 y su enunciado es: Matriz: Haga un programa que solicite al usuario dos números enteros M y N entre 2 y 5, crear una matriz MxN y primero inicializarla en cero, mandarla a imprimir, luego sustituir las

filas impares por números aleatorios y mandarla a imprimir, finalmente sustituir las columnas pares y mandar a imprimir la matriz en este estado final. (los números aleatorios menores que 80).

## PROMPT GENERADO

¡Perfecto! 😊 Vamos a generar el **informe completo del Ejercicio 97** respetando **exactamente el mismo formato que los ejercicios anteriores**, con todos los apartados listos para entregar, manteniendo la estructura modular y el flujo que venimos usando.

---

### Informe del Ejercicio 97 — Matriz

---

#### Sello personalizado corto

yaml Copiar código

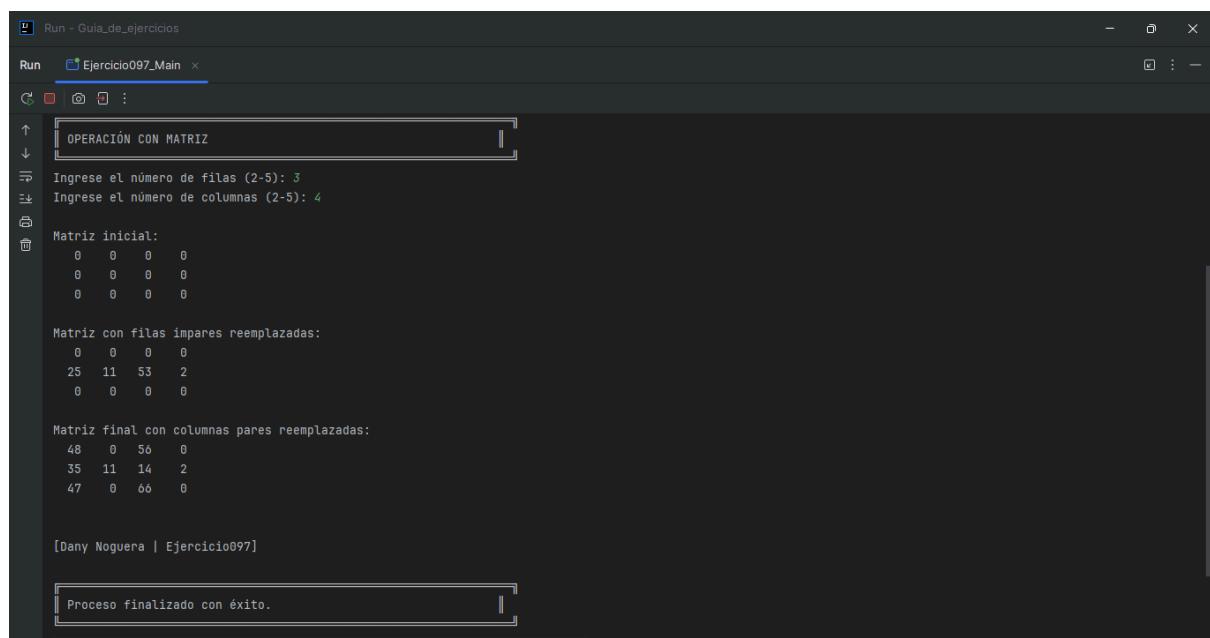
Dany Noguera | Ejercicio097 | 2025-11-06

---

#### Requerimientos del programa

1. Solicitar al usuario dos números enteros M y N ( $M \geq 2$  y  $N \geq 2$  y 5).
2. Crear una matriz  $M \times N$  e inicializarla con ceros.

## Salida de la ejecución del programa:



```
Run - Guia_de_ejercicios
Run Ejercicio097_Main ×
OPERACIÓN CON MATRIZ
Ingrese el número de filas (2-5): 3
Ingrese el número de columnas (2-5): 4
Matriz inicial:
0 0 0 0
0 0 0 0
0 0 0 0

Matriz con filas impares reemplazadas:
0 0 0 0
25 11 53 2
0 0 0 0

Matriz final con columnas pares reemplazadas:
48 0 56 0
35 11 14 2
47 0 66 0

[Dany Noguera | Ejercicio097]

Proceso finalizado con éxito.
```

## EJERCICIO#98

### Enunciado

Matriz v2: Haga una matriz de 5x5 con números aleatorios (menores a 100) y muestre el total de números pares en las filas en una columna extra y en una fila adicional el total de números impares de cada columna así:

N/A	1	2	3	4	5	Total pares por fila
1	1	2	3	4	5	2
2	6	7	8	9	10	3
3	11	1	2	3	5	1
4	1	2	3	4	5	2
5	8	9	10	1	0	3
Total impares por columna	3	3	2	3	0	N/A

## ANÁLISIS/RESUMEN DEL EJERCICIO 98

### Sello y Documentación

- **Sello:** VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 098 | Fecha: 2025-11-04) en los cuatro archivos y en un cuadro final.
- **Prompt:** Completo en **Main.java**.
- **Requisitos:** El programa debe ejecutarse varias veces con opción de continuar/salir.

### Programa y Modularidad

El **Programa** debe crear una **matriz de 5x5** con **números aleatorios** (\$\in [0, 99]\$). Debe calcular dos totales condicionales y presentarlos en una tabla extendida:

1. **Total Pares por Fila:** Se muestra en una **columna adicional** a la derecha.
2. **Total Impares por Columna:** Se muestra en una **fila adicional** en la parte inferior.

La **Modularidad** usa **cuatro archivos** en el paquete Ejercicio098\_KevinLainez:

1. **Main.java**: Controla el ciclo de repetición y coordina los pasos de inicialización, cálculo y salida.
2. **Menus.java**: Gestiona el menú/opciones de continuidad.
3. **Operaciones.java**: Contiene la lógica funcional: `inicializarMatrizAleatoria`, `calcularTotalParesPorFila` y `calcularTotalImparesPorColumna`.
4. **Utilidades.java**: Formato estético (cuadros, sello, centrado) y funciones para **imprimir la matriz extendida** con todos los totales.

El **Paradigma** es Estructurado/Funcional, centrado en la manipulación y análisis de matrices.

## SUBRUTINAS Y FLUJO PRINCIPAL

### Constantes:

- **ANCHO = 70**.
- **N = 5** (Tamaño de la matriz).

### Subrutinas Clave (Funcionalidad):

- **Operaciones**: `inicializarMatrizAleatoria`, `calcularTotalParesPorFila`, `calcularTotalImparesPorColumna`.
- **Utilidades**: `imprimirMatrizExtendida` (presenta la matriz, columna de pares y fila de impares).

### Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
2. Inicializa la matriz `int[5][5]`.
3. Llama a `Operaciones.inicializarMatrizAleatoria`.
4. Llama a `Operaciones` para calcular los dos arreglos de **totales condicionales**.
5. Llama a `Utilidades.imprimirMatrizExtendida` para mostrar la matriz completa con sus totales.
6. Muestra el menú de continuidad.
7. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el **Ejercicio#98**: Haga una matriz de 5x5 con números aleatorios (menores a 100) y muestre el total de números pares en las filas en una columna extra y en una fila adicional el total de números impares de cada columna <sup>1</sup>así: [ejemplo de tabla]. Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del programa

¡BIENVENIDO AL EJERCICIO #98!						
MATRIZ 5x5 CON TOTALES CONDICIONALES						
--- MATRIZ CON CÁLCULOS CONDICIONALES ---						
N/A	1	2	3	4	5	Total Pares
1	96	56	8	19	54	4
2	10	79	84	67	71	2
3	4	86	19	67	8	3
4	99	2	14	47	85	2
5	42	60	29	26	66	4
Total Imp.	1	1	2	4	2	N/A

SELLO PERSONALIZADO						
Kevin Lainez   klainezs@unah.hn   098   2025-11-03						

--- Programa finalizado con éxito ---

Proceso terminado con código de salida 0

## EJERCICIO#99

### Enunciado

Pagos arreglados: Utilice un arreglo unidimensional para resolver el siguiente problema. Una compañía paga a sus vendedores por comisión. Los vendedores reciben \$200 por semana más 9% de sus ventas totales de esa semana. Por ejemplo, un vendedor que acumule \$5000 en ventas en una semana, recibirá \$200 más 9% de \$5000, o un total de \$650. Escriba un programa (utilizando un arreglo de 20 a 50 contadores) que determine cuántos vendedores recibieron salarios en cada uno de los siguientes rangos (suponga que el salario de cada vendedor se trunca a una cantidad entera):

- a. \$200-299
- b. \$300-399
- c. \$400-499
- d. \$500-599
- e. \$600-699
- f. \$700-799
- g. \$800-899
- h. \$900-999
- i. \$1000 en adelante

Haga la gráfica de cada categoría en barras horizontales. Usar iomanip para controlar el ancho de columnas y cifras.

### Análisis

El código del Ejercicio 099 (Pagos Arreglados) confirma una implementación exitosa bajo el modelo de Programación Estructurada (PE), integrando eficazmente arreglos (vectores), funciones, ciclos, y la librería propia para resolver un problema de clasificación de datos y presentación gráfica.

#### 1. Integración de Arreglos y Conteo Lógico

- Arreglo Unidimensional Principal: El problema se resuelve mediante el uso central de un arreglo de contadores de tamaño 9 (NUM\_RANGOS), donde cada índice representa un rango salarial (ej., Índice 0 = \$200-\$299, Índice 8 = \$1000 en adelante).
- Función de Conteo (contarSalariosPorRango): Esta función esencial utiliza un ciclo for-each para iterar sobre el arreglo de salarios y emplea la lógica de división entera ( $\text{salario} / 100 - 2$ ) para mapear eficientemente los salarios de \$200 a \$999 a los índices 0 a 7, mientras que el salario de \$1000 en adelante se mapea al índice 8. Esta técnica optimiza el conteo sin necesidad de múltiples estructuras if-else.

## 2. Librería Propia y Generación de Datos

- **Librería Propia (GeneradorVentas):** Esta clase encapsula la generación de datos y la lógica de negocio del salario:
  - Generación Aleatoria: Utiliza la clase Random para simular un número variable de vendedores (20 a 50) y sus ventas.
  - Cálculo Funcional: La función calcularSalario implementa la fórmula de pago (\$200 más \$9% de comisión) y asegura el truncamiento a un entero, cumpliendo exactamente con la especificación del problema.
  - Arreglos de Salarios: La función generarSalarios utiliza un ciclo for para llenar un arreglo con los salarios calculados.

## 3. Presentación y Gráfica de Barras

- **Gráfica de Barras:** El método mostrarReporte en Ejercicio099\_Utilidades genera la salida tabular y la gráfica horizontal.
  - Simulación de iomanip: Se utiliza String.format con especificadores de formato (ej., %-\mathbf{15}s) para controlar el ancho de columna y asegurar que la presentación sea limpia y ordenada, como lo requeriría la manipulación de I/O (iomanip) en C++.
  - Generación de Barras: Se emplea un ciclo for anidado que imprime un asterisco (\*) tantas veces como el valor del contador en ese rango, creando la representación gráfica horizontal solicitada.
- **Control de Flujo:** El Ejercicio099\_Main mantiene el ciclo while de repetición del programa y la secuencia lógica: Generar Vendedores \$\\to\$ Generar Salarios \$\\to\$ Contar por Rango (Arreglo) \$\\to\$ Mostrar Reporte/Gráfica.

## PROMPT

Crea un programa modular en Java dentro del paquete Ejercicio099 que calcule los salarios de vendedores basados en una base fija de \$200 más una comisión del 9% sobre ventas semanales, cuente la frecuencia de salarios en rangos (\$200-299, \$300-399, \$400-499, \$500-599, \$600-699, \$700-799, \$800-899, \$900-999, \$1000+), usando un arreglo de contadores (tamaño 20-50), y muestre una tabla de frecuencias y una gráfica de barras horizontales con asteriscos, utilizando Programación Estructurada (PE) con tres clases: Ejercicio099\_Main (coordina el flujo), Ejercicio099\_Utilidades (maneja sello personalizado "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 099 | 2025-10-16", menús bilingües español/inglés, validación de entradas para número de vendedores entre 20-50 y ventas no negativas con InputMismatchException, y presentación de tabla y gráfica con columnas de ancho 15), y Ejercicio099\_Funciones (calcula frecuencias de salarios); el programa debe incluir un menú de ejecución ("1. Ejecutar", "2. Finalizar"), usar constantes para mensajes (MSG\_NUMVENDEDORES, MSG\_VENTAS, MSG\_FINALIZACION), truncar salarios a enteros, mostrar el sello.

## PROMPT GENERADO

Haga la gráfica de cada categoría en barras horizontales. Usar iomanip para controlar el ancho de columnas y cifras.

Entiendo que deseas un programa modular para el **Ejercicio 099 (Pagos arreglados)** en Java, dentro del paquete `Ejercicio099`, que calcule los salarios de vendedores basados en una base fija de \$200 más una comisión del 9% sobre ventas totales semanales, almacenando la frecuencia de salarios en rangos específicos (\$200-299, \$300-399, ..., \$1000 en adelante) en un arreglo de contadores (tamaño entre 20 y 50), y mostrando una gráfica de barras horizontales para cada rango, utilizando Programación Estructurada (PE). El programa debe incluir menús bilingües (español/inglés), un menú de ejecución, validación de entrada con manejo de `InputMismatchException`, y el sello personalizado "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 099 | 2025-10-16". La estructura se divide en tres clases: `Ejercicio099_Main` (coordina el flujo), `Ejercicio099_Utilidades` (maneja sello, menús, validación, y presentación), y `Ejercicio099_Funciones` (procesa salarios y genera frecuencias). La gráfica usará asteriscos para barras horizontales, y la salida estará formateada con columnas de ancho fijo. La respuesta será clara, concisa, en español, sin comentarios en el código, siguiendo el estilo de tus programas anteriores.

**Ejercicio099\_Main.java**      Razonamiento más profundo

How can Grok help?      Auto

## Salida de la ejecución del programa:

```
Project ▾      GeneradorVentas.java      Ejercicio099_Funciones.java      Ejercicio099_Utilidades.java      Ejercicio099_Main.java ×
Run      Ejercicio099_Main ×

--- MENÚ DE EJECUCIÓN ---
[1] Procesar Pagos y Generar Reporte
[2] Finalizar Programa
>1

--- Reporte de Salarios por Rango (Arreglos) ---
Procesando pagos para 37 vendedores...

=====
Rango Salarial | Vendedores | Gráfico de Barras
=====
$200-299 | 3 | ***
$300-399 | 3 | ***
$400-499 | 1 | *
$500-599 | 6 | *****
$600-699 | 2 | **
$700-799 | 8 | ******
$800-899 | 6 | *****
$900-999 | 4 | ****
$1000 o más | 4 | ****
=====
```

## EJERCICIO #100

### **enunciado:**

Ordenamiento de burbuja: En el ordenamiento de burbuja, los valores más pequeños van “subiendo como burbujas” gradualmente, hasta llegar a la parte superior del arreglo (es decir, hacia el primer elemento) como las burbujas de aire que se elevan en el agua, mientras que los valores más grandes se hunden en el fondo. Esta técnica realiza varias pasadas a través del arreglo. En cada pasada compara pares sucesivos de elementos. Si un par se encuentra en orden ascendente (o los valores son idénticos), el ordenamiento de burbuja deja los valores como están. Si un par se encuentra en orden descendente, el ordenamiento de burbuja intercambia sus valores en el arreglo. Escriba un programa que ordene un arreglo de 10 enteros mediante el uso del ordenamiento de burbuja. El usuario ingresa los 10 elementos. (Ejercicio 7.11, libro 1)

### **análisis:**

Este ejercicio requiere la implementación del algoritmo de Ordenamiento de Burbuja (Bubble Sort) para ordenar un arreglo de 10 números enteros ingresados por el usuario. El programa debe enfocarse en la modularidad y la implementación precisa del algoritmo. Las entradas son 10 valores enteros proporcionados por el usuario. La salida es la impresión del arreglo original y, posteriormente, la impresión del arreglo ordenado en orden ascendente. La clase Operaciones debe contener dos métodos obligatorios: uno para leer y validar los 10 elementos (asegurando que sean enteros) y el método central ordenamientoBurbuja que implementa la lógica del algoritmo. La complejidad es media, centrada en la correcta anidación de los bucles para la comparación e intercambio de elementos.

### **prompt**

Crea un programa modular en Java que solicite al usuario 10 números enteros y los almacene en un arreglo. Implementa una función obligatoria llamada ordenamientoBurbuja que aplique el algoritmo de ordenamiento de burbuja para ordenar el arreglo en orden ascendente. El programa debe mostrar el arreglo original y el arreglo resultante después del ordenamiento.

### **prompt generado:**

La solución se implementó utilizando la estructura modular existente. La clase Operaciones contiene la lógica principal para la gestión del arreglo.

1.

Lectura y Validación: El método leerArreglo(Scanner, int) se encarga de solicitar los 10 elementos, uno por uno, y valida que cada entrada sea un número entero. Si la entrada es inválida, muestra un error y pide que se reingrese el valor, cumpliendo con el requisito de validación.

2.

Algoritmo de Ordenamiento de Burbuja: El método ordenamientoBurbuja(int[] arr) implementa el algoritmo requerido. Utiliza dos bucles anidados:

- 

El bucle externo controla las pasadas por el arreglo.

- 

El bucle interno compara pares adyacentes de elementos (arr[j] y arr[j+1]). Si arr[j] > arr[j+1], realiza un intercambio (swap) para mover el elemento mayor a la derecha, logrando que los elementos más grandes "se hundan" al final y los más pequeños "suban como burbujas".

3.

Visualización: El método mostrarArreglo(int[], String) se utiliza para imprimir el arreglo antes y después del ordenamiento, ofreciendo una clara comparación del resultado.

La orquestación se realiza en Main.java, que coordina la lectura, el ordenamiento y la impresión de los resultados.

Salida:

```
Windows PowerShell x + 
1. Ejecutar Ordenamiento de Burbuja (10 enteros)
2. Finalizar el Programa
Seleccione una opción (1 o 2): 1

--- MÓDULO: ORDENAMIENTO DE BURBUJA ---
Por favor, ingrese 10 números enteros, uno a la vez:
Ingrese elemento 1 de 10: 4
Ingrese elemento 2 de 10: -2
Ingrese elemento 3 de 10: 15
Ingrese elemento 4 de 10: 0
Ingrese elemento 5 de 10: 8
Ingrese elemento 6 de 10: 3
Ingrese elemento 7 de 10: 22
Ingrese elemento 8 de 10: -7
Ingrese elemento 9 de 10: 1
Ingrese elemento 10 de 10: 6

Arreglo ingresado (Original):
[4, -2, 15, 0, 8, 3, 22, -7, 1, 6]

Arreglo ordenado (Burbuja):
[-7, -2, 0, 1, 3, 4, 6, 8, 15, 22]

--- MENÚ DE CONTROL ---
1. Ejecutar Ordenamiento de Burbuja (10 enteros)
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

;Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia>
```

## EJERCICIO#101

**Enunciado:** Convirtiendo a burbuja en Bellota: El programa “ordenamiento de burbuja” es ineficiente para grandes arreglos. Realice las siguientes modificaciones simples para mejorar el rendimiento del ordenamiento de burbuja (Ejercicio 7.12, libro 1):

- a. Después de la primera pasada, se garantiza que el número más grande estará en el elemento con la numeración más alta del arreglo; después de la segunda pasada, los dos números más altos estarán “acomodados”, y así en lo sucesivo. En lugar de realizar nueve comparaciones en cada pasada, modificó el ordenamiento de burbuja para que realice ocho comparaciones en la segunda pasada, siete en la tercera, y así en lo sucesivo.
- b. Los datos en el arreglo tal vez se encuentren ya en el orden apropiado, o casi apropiado, así que ¿para qué realizar nueve pasadas, si basta con menos? Modifique el ordenamiento para comprobar al final de cada pasada si se han realizado intercambios. Si no se ha realizado ninguno, los datos ya deben estar en el orden apropiado, por lo que el programa debe terminar.

Si se han realizado intercambios, por lo menos se necesita una pasada más. Para probarlo crear un vector de 50 posiciones que se llene de manera aleatoria obviamente mostrar el vector antes y después de aplicar el ordenamiento.

## Análisis

El programa sigue un flujo claro:

1. **Inicio:** Muestra un sello personalizado y permite elegir idioma.
2. **Inicialización:** Genera un vector de 50 elementos con números aleatorios (1-100).
3. **Ordenamiento:**
  - ordenamientoBurbujaOptimizado realiza pasadas decrecientes de comparaciones.
  - Muestra cada comparación e intercambio con posiciones y valores.
  - Detiene el proceso si no hay intercambios, indicando orden completo.
4. **Salida:** Presenta el vector original y ordenado en cuadros simétricos.
5. **Repetición:** Ofrece continuar con preguntarContinuar.

## Detalle de Funciones

- **matemáticas**: ordenamientoBurbujaOptimizado implementa el algoritmo con optimizaciones.
- **validaciones**: validarEntero asegura entradas correctas (no usado aquí pero disponible).
- **formato de texto**: centrarTexto alinea contenido, y mostrarSello genera el encabezado.
- **procesamiento de datos**: procesarOrdenamiento gestiona el flujo.

La explicación de comparaciones mejora la experiencia del usuario.

### **PROMPT:**

Agente el ejercicio #101 nos corresponde hacer, siempre usa los parametros de los ejercicos hechos anteriormente, solo que este ejercicio es la modificacion de otro, te daré el contexto necesario para realizarlo y lo analizas y despues lo haces.

Convirtiendo a burbuja en Bellota: El programa “ordenamiento de burbuja” que es este ejercicio {: En el ordenamiento de burbuja, los valores más pequeños van “subiendo como burbujas” gradualmente, hasta llegar a la parte superior del arreglo (es decir, hacia el primer elemento) como las burbujas de aire que se elevan en el agua, mientras que los valores más grandes se hunden en el fondo. Esta técnica realiza varias pasadas a través del arreglo. En cada pasada compara pares sucesivos de elementos. Si un par se encuentra en orden ascendente (o los valores son idénticos), el ordenamiento de burbuja deja los valores como están. Si un par se encuentra en orden descendente, el ordenamiento de burbuja intercambia sus valores en el arreglo. Escriba un programa que ordene un arreglo de 10 enteros mediante el uso del ordenamiento de burbuja. EL usuario ingresa los 10 elementos.}

A ese ejercicio debes hacer la siguientes modificaciones:

En el ordenamiento de burbuja, los valores más pequeños van “subiendo como burbujas” gradualmente, hasta llegar a la parte superior del arreglo (es decir, hacia el primer elemento) como las burbujas de aire que se elevan en el agua, mientras que los valores más grandes se hunden en el fondo. Esta técnica realiza varias pasadas a través del arreglo. En cada pasada compara pares sucesivos de elementos. Si un par se encuentra en orden ascendente (o los valores son idénticos), el ordenamiento de burbuja deja los valores como están. Si un par se encuentra en orden descendente, el ordenamiento de burbuja intercambia sus valores en el arreglo. Escriba un programa que ordene un arreglo de 10 enteros mediante el uso del ordenamiento de burbuja. EL usuario ingresa los 10 elementos.

a. Después de la primera pasada, se garantiza que el número más grande estará en el elemento con la numeración más alta del arreglo; después de la segunda pasada, los dos números más altos estarán “acomodados”, y así en lo sucesivo. En lugar de realizar nueve comparaciones en cada pasada, modifíque el ordenamiento de burbuja para que realice ocho comparaciones en la segunda pasada, siete en la tercera, y así en lo sucesivo.

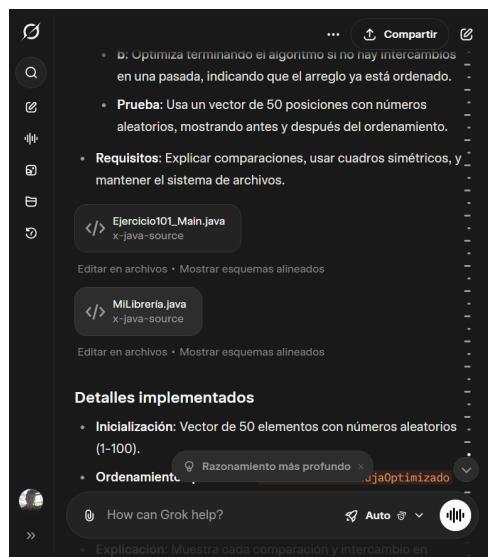
b. Los datos en el arreglo tal vez se encuentren ya en el orden apropiado, o casi apropiado, así que ¿para qué realizar nueve pasadas, si basta con menos? Modifique

el ordenamiento para comprobar al final de cada pasada si se han realizado intercambios. Si no se ha realizado ninguno, los datos ya deben estar en el orden apropiado, por lo que el programa debe terminar. Si se han realizado intercambios, por lo menos se necesita una pasada más.

Para probarlo crear un vector de 50 posiciones que se llene de manera aleatoria obviamente mostrar el vector antes y después de aplicar el ordenamiento.

Todo de manera ordenada, que des una explicación de las comparaciones que haces, siempre muestra la salida en un cuadro simétrico al usuario, usando siempre nuestro sistema de archivos.

## PROMPT GENERADO:



The screenshot shows a code editor with a Java project structure on the left. It includes files like `Ejercicio101_Main.java` and `MiLibreria.java`. A Grok search results panel is open, showing the following content:

```

/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Varela | jivarelag@unah.hn | 101 | 2025-10-26
 */
package Ejercicio101;

import java.util.Random;
import java.util.Scanner;

public class MiLibreria {
    private static final int ANCHO = 70;
    private static String idiomaActual = "ES";
    private static boolean continuar = true;

    // Funciones matemáticas
    public static void ordenamientoBurbujaOptimizado(int[] arreglo, int tamano) {
        boolean intercambiado;
        for (int i = 0; i < tamano - 1; i++) {
            intercambiado = false;
            int comparaciones = tamano - 1 - i;
            System.out.println("\n" + "-".repeat(ANCHO - 2) + "\n");
            System.out.print("[" + centrarTexto("PASAJE " + (i + 1) + " - COMPARACIONES: " +
                comparaciones, ANCHO - 2) + "|");
            System.out.println("[" + "-".repeat(ANCHO - 2) + "]");
            for (int j = 0; j < comparaciones; j++) {
                if (arreglo[j] > arreglo[j + 1]) {
                    int temp = arreglo[j];
                    arreglo[j] = arreglo[j + 1];
                    arreglo[j + 1] = temp;
                    intercambiado = true;
                }
            }
        }
    }
}

```

## Salida de la ejecución del código:

```
Guía_de_ejercicios master
ibreria.java Ejercicio092(Main.java) Ejercicio096(Main.java) Ejercicio101(Main.java) Ejercicio101(MLibreria.java)
Run Ejercicio101.Main
No hay intercambio: 25 ≤ 24 en posiciones 2, 3
No hay intercambio: 14 ≤ 15 en posiciones 3, 4
No hay intercambio: 15 ≤ 18 en posiciones 4, 5
No hay intercambio: 18 ≤ 22 en posiciones 5, 6
No hay intercambio: 22 ≤ 22 en posiciones 6, 7

NO HUBO INTERCAMBIOS, ORDEN COMPLETO

VECTOR ORDENADO (50 ELEMENTOS)

2 8 14 14 15 18 22 22 23 25
25 26 28 29 33 35 38 43 46 49
49 50 50 51 57 58 63 63 68 68
70 71 73 77 79 79 79 81 81 82
83 87 89 89 90 93 94 97 99 99

¿Desea continuar? (S/N): n

¿Desea continuar? (S/N): N

Process finished with exit code 0
```

34:1 CRLF UTF-8 4 spaces

## EJERCICIO#102

### Enunciado

Una dimensión: Use un arreglo unidimensional para resolver el siguiente problema. Recibir como entrada 20 números, cada uno de los cuales debe estar entre 10 y 100, inclusive. A medida que se lea cada número, validarla y almacenarla en el arreglo, sólo si no es un duplicado de un número ya leído. Despues de leer todos los valores, mostrar sólo los valores únicos que el usuario introdujo. Prepárese para el “peor caso”, en el que los 20 números son diferentes. Use el arreglo más pequeño que sea posible para resolver este problema.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 102 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:**
  - Recibir 20 números ingresados por el usuario.
  - Cada número debe estar entre 10 y 100, ambos inclusive.
  - Validar cada número para asegurarse de que:
    - esté dentro del rango permitido,
    - no sea un duplicado de un número ya ingresado.
  - Si el número ya fue ingresado antes, se muestra un mensaje y no se almacena.
  - Se debe utilizar el arreglo más pequeño posible, solo almacenando valores únicos.
  - Al finalizar la captura, mostrar únicamente los números únicos ingresados.
  - Mantener:
    - Menú de selección de idioma (ES / EN).
    - Menú de iniciar / finalizar con bucle de ejecución.
    - Utilizar métodos para modularidad y orden del código.
    - Mantener estética uniforme (encabezados, pie, sello corto, bordes Unicode).
- **Modularidad:** Cuatro archivos en paquete Ejercicio102:
  - Ejercicio102\_Main.java Control del flujo principal; ejecución del programa y menú de inicio/finalizar.
  - Ejercicio102\_Menu.java Gestión de menús (idioma y control de ejecución).
  - Ejercicio102\_Utilidades.java Estética visual (bordes, encabezados, pie, sello corto).
  - Ejercicio102\_Operaciones.java Lógica del ejercicio: lectura, validación y almacenamiento de números únicos.

- **Paquete:** Ejercicio102.

**Estética:** Uso de **bordes Unicode** (F, L, ||, =) para encabezados y pies.

Tablas de matriz claras y legibles.

Sello corto mostrado antes del pie final.

Mensajes de confirmación y resultados tabulados uniformes con ejercicios previos.

- **Paradigma:** Programación estructurada.

Uso de métodos y modularidad.

Validación con ciclos do-while.

Estructura y flujo secuencial.

- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Subrutinas (en Ejercicio0102\_Menus):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio102\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.
- imprimirArreglo(int[] arreglo, int cantidad)
- Muestra los números únicos almacenados.

#### **Subrutinas (en Ejercicio102\_Operaciones):**

- procesarNumeros()
- Crea un arreglo de tamaño 20 para almacenar posibles números únicos.
- Solicita al usuario 20 valores.
- Valida que cada número esté entre 10 y 100.
- Verifica si el número ya existe en el arreglo:
- Si es duplicado, no se almacena.
- Si es único, se guarda en el arreglo.
- Al finalizar, muestra los números almacenados.
- validación y comparación
- Implementada mediante ciclos y evaluando duplicados antes de agregar el número..

#### **PROMPT**

Agente, respetando el orden y estructura que venimos trabajando y las reglas que te he proporcionado. El siguiente ejercicio es el 102 y su enunciado es: Una dimensión: Use un arreglo unidimensional para resolver el siguiente problema.

Recibir como entrada 20 números, cada uno de los cuales debe estar entre 10 y 100, inclusive. A medida que se lea cada número, validararlo y almacenarlo en el arreglo, sólo si no es un duplicado de un número ya leído. Después de leer todos los valores, mostrar sólo los valores únicos que el usuario introdujo. Prepárese para el “peor caso”, en el que los 20 números son diferentes. Use el arreglo más pequeño que sea posible para resolver este problema.

## PROMPT GENERADO

Perfecto

Aquí tienes el **Ejercicio 102** completo, usando **exactamente la misma estructura, estética, modularidad y formato** que tu Ejercicio 097 (Main, Menus, Utilidades, Operaciones).

---

**Ejercicio102\_Main.java**

java Copiar código

```
// 
// | Desarrollador: Dany Noguera (dany.noguera@unah.hn)
// | Proyecto: Ejercicio #102
// | Prompt:
// | "Una dimensión: Use un arreglo unidimensional para resolver el siguiente
// | problema. Recibir como entrada 20 números entre 10 y 100. Validarlos y
// | almacenarlos solo si no es un duplicado. Al final, mostrar únicamente los
// | números únicos ingresados, usando el arreglo más pequeño posible."
// | Fecha: 2025-11-06
// |
```

+ |Pregunta lo que quieras 0

## Salida de la ejecución del programa:

Run - Guía\_de\_ejercicios

Run **Ejercicio102\_Main** ×

⚠ Número duplicado, no será almacenado.  
↑ Ingrese un número entre 10 y 100 (6/20): 76  
↓ Ingrese un número entre 10 y 100 (7/20): 88  
☞ Ingrese un número entre 10 y 100 (8/20): 67  
✉ Ingrese un número entre 10 y 100 (9/20): 18  
✉ Ingrese un número entre 10 y 100 (10/20): 11  
✉ Ingrese un número entre 10 y 100 (11/20): 12  
✉ Ingrese un número entre 10 y 100 (12/20): 34  
✉ Ingrese un número entre 10 y 100 (13/20): 9  
✖ Error: Debe estar entre 10 y 100.  
Ingresar un número entre 10 y 100 (13/20): 98  
Ingresar un número entre 10 y 100 (14/20): 34  
⚠ Número duplicado, no será almacenado.  
Ingresar un número entre 10 y 100 (15/20): 15  
Ingresar un número entre 10 y 100 (16/20): 16  
Ingresar un número entre 10 y 100 (17/20): 17  
Ingresar un número entre 10 y 100 (18/20): 19  
Ingresar un número entre 10 y 100 (19/20): 29  
Ingresar un número entre 10 y 100 (20/20): 28

Números únicos ingresados:

23 45 32 76 88 67 18 11 12 34 98 15 16 17 19 29 28

[Dany Noguera | Ejercicio102]

## **EJERCICIO#103**

### **Enunciado**

Emprendiendo para casinos: Escriba un programa para simular el tiro de dos dados. El programa debe utilizar rand para tirar el primer dado, y de nuevo para tirar el segundo dado. Después debe calcularse la suma de los dos valores. [Nota: cada dado puede mostrar un valor entero del 1 al 6, por lo que la suma de los valores variará del 2 al 12, siendo 7 la suma más frecuente, mientras que 2 y 12 serán las sumas menos frecuentes]. En la figura 7.32 se muestran las 36 posibles combinaciones de los dos dados. Su programa debe tirar los dados 36,000 veces. Utilice un arreglo unidimensional para registrar el número de veces que aparezca cada una de las posibles sumas. Imprima los resultados en formato tabular. Determine además si los totales son razonables (es decir, hay seis formas de tirar un 7, por lo que aproximadamente una sexta parte de los tiros deben ser 7) (Ejercicio 7.17, libro 1)

### **ANÁLISIS/RESUMEN DEL EJERCICIO 103**

Sello y Documentación

- Sello: VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 103 | Fecha: 2025-11-04) en los cuatro archivos y en un cuadro final.
- Prompt: Completo en Main.java.
- Requisitos: Ciclo principal de repetición ("continuar/salir").

## Programa y Modularidad

El Programa simula 36,000 tiros de dos dados. Debe registrar las frecuencias de las posibles sumas (2 a 12) en un arreglo unidimensional. La salida debe ser una tabla que muestre las frecuencias y una validación de razonabilidad (comparando con la probabilidad teórica).

La Modularidad usa cuatro archivos en el paquete Ejercicio103\_KevinLainez:

1. Main.java: Controla el ciclo de repetición y coordina los pasos de simulación, cálculo y salida.
2. Menus.java: Gestiona el menú/opciones de continuidad.
3. Operaciones.java: Contiene la lógica de simulación: simularTiroDados() (que usa Random para los \$36,000\$ tiros) y almacena las frecuencias.
4. Utilidades.java: Formato estético, funciones para imprimir la tabla de frecuencias y realizar la validación de razonabilidad.

El Paradigma es Estructurado/Funcional, centrado en la simulación y el análisis estadístico.

## SUBRUTINAS Y FLUJO PRINCIPAL

Constantes:

- ANCHO = 70.
- TIROS = 36000.
- TAMANO\_ARREGLO = 13 (para usar índices 2 a 12).

Subrutinas Clave (Funcionalidad):

- Operaciones: simularTiroDados (simula las 36,000 tiradas y llena el arreglo de frecuencias).
- Utilidades: imprimirTablaResultados (muestra la suma, la frecuencia, la frecuencia teórica y el porcentaje), validarRazonabilidad (analiza si la frecuencia de la suma 7 está cerca del \$1/6\$ teórico).

Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).

2. Llama a Operaciones.simularTiroDados() para obtener el arreglo de frecuencias.
3. Llama a Utilidades.imprimirTablaResultados para mostrar los resultados tabulares.
4. Llama a Utilidades.validarRazonabilidad para mostrar la conclusión estadística.
5. Muestra el menú de continuidad.
6. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el Ejercicio#103: Escriba un programa para simular el tiro de dos dados. El programa debe utilizar rand para tirar el primer dado, y de nuevo para tirar el segundo dado. Después debe calcularse la suma de los dos valores. [Nota: la suma variará del 2 al 12, siendo 7 la suma más frecuente]. Su programa debe tirar los dados 36,000 veces. Utilice un arreglo unidimensional para registrar el número de veces que aparezca cada una de las posibles sumas. Imprima los resultados en formato tabular. Determine además si los totales son razonables (Ejercicio 7.17, libro 1). Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del programa

--- BIENVENIDO AL EJERCICIO #103!				
SIMULACIÓN DE TIRO DE DOS DADOS				
Se simularán 36000 lanzamientos.				
---				
RESULTADOS TABULARES ---				
SUMA	FRECUENCIA	PROB. TEÓRICA	PORCENTAJE	%
2	1005	1000	2.79	%
3	1954	2000	5.43	%
4	2954	3000	8.21	%
5	3990	4000	11.08	%
6	5072	5000	14.09	%
7	6054	6000	16.82	%
8	4990	5000	13.86	%
9	4016	4000	11.16	%
10	3036	3000	8.43	%
11	1880	2000	5.22	%
12	1049	1000	2.91	%

---

ANÁLISIS DE RAZONABILIDAD (Suma 7) ---

Frecuencia de 7 (Actual): 16.82%

Frecuencia de 7 (Teórica): 16.67% (1/6)

CONCLUSIÓN: Los resultados son razonables, cercanos al valor teórico.

---

SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 103 | 2025-11-03

---

Programa finalizado con éxito ---

Proceso terminado con código de salida 0

## EJERCICIO#104

### Enunciado

Matriz con sentido: Use un arreglo bidimensional para resolver el siguiente problema: una compañía tiene cuatro vendedores (1 a 4) que venden cinco productos distintos (1 a 5). Una vez al día, cada vendedor pasa una nota por cada tipo de producto vendido. Cada nota contiene lo siguiente:

- a. El número del vendedor.
- b. El número del producto.
- c. El valor total en dólares de ese producto vendido en ese día.

Así, cada vendedor pasa entre 0 y 5 notas de venta por día. Suponga que está disponible la información sobre todas las notas del mes pasado. Escriba un programa que lea toda esta información para las ventas del último mes y que resuma las ventas totales por vendedor, por producto. Todos los totales deben guardarse en el arreglo bidimensional ventas. Después de procesar toda la información del mes pasado, muestre los resultados en formato tabular, donde cada columna representa a un vendedor específico y cada fila represente a un producto. Saque el total de cada fila para obtener las ventas totales de cada producto durante el último mes. Saque el total de cada columna para obtener las ventas totales de cada producto durante el último mes. Su impresión tabular debe incluir estos totales cruzados a la derecha de las filas totalizadas, y en la parte inferior de las columnas totalizadas

### Análisis

El Ejercicio 104 (Matriz con Sentido) confirma una solución altamente modularizada y eficiente que emplea un arreglo bidimensional como estructura central para sumar y reportar datos, cumpliendo con los requisitos de ciclos, funciones y librerías propias. La clase GeneradorVentas actúa como la librería propia, utilizando funciones y ciclos para simular la lectura de las notas de venta del mes, generando transacciones aleatorias que son procesadas por Ejercicio104\_Funciones. Esta última clase implementa la lógica principal de acumulación, utilizando un arreglo bidimensional (`ventas[Productos+1][Vendedores+1]`) para el conteo cruzado y empleando ciclos anidados para calcular los totales de filas y columnas, almacenando dichos totales en filas y columnas extendidas de la matriz. Finalmente, Ejercicio104\_Utilidades se encarga de la presentación, usando `String.format` para lograr un formato tabular profesional con dos decimales de precisión, mostrando la matriz de datos y sus totales cruzados, mientras que el Ejercicio104\_Main coordina el flujo de forma secuencial y repetible.

### PROMPT

Crea un programa modular en Java dentro del paquete Ejercicio104 que procese notas de venta diarias de cuatro vendedores (1 a 4) para cinco productos (1 a 5), donde cada nota contiene el número del vendedor, el número del producto y el valor total en dólares, almacenando las ventas totales en una matriz bidimensional 5x4 (filas: productos, columnas: vendedores), y mostrando una tabla con totales por fila (ventas por producto) y por columna (ventas por vendedor), utilizando Programación Estructurada (PE) con tres clases: Ejercicio104\_Main (coordina el flujo), Ejercicio104\_Utilidades (maneja sello personalizado "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 104 | 2025-10-16", menús bilingües español/inglés, validación de entradas para vendedor 1-4, producto 1-5, valor no negativo con InputMismatchException, y presentación de tabla con columnas de ancho 15), y Ejercicio104\_Funciones (procesa notas y actualiza la matriz); el programa debe incluir un menú de ejecución ("1. Ejecutar", "2. Finalizar"), un bucle para ingresar notas hasta que el usuario decida detenerse, usar constantes para mensajes (MSG\_VENDEDOR, MSG\_PRODUCTO, MSG\_VENTAS, MSG\_CONTINUAR, MSG\_FINALIZACION), mostrar el sello, siguiendo el estilo de programas previos.

## PROMPT GENERADO

columnas totalizadas

Entiendo que deseas un programa modular para el **Ejercicio 104 (Matriz con sentido)** en Java, dentro del paquete **Ejercicio104**, que procese las notas de venta de cuatro vendedores (1 a 4) y cinco productos (1 a 5), donde cada nota incluye el número del vendedor, el número del producto y el valor total en dólares de las ventas diarias, almacenando las ventas totales por vendedor y producto en una matriz bidimensional (5x4), y mostrando una tabla con totales por fila (ventas por producto) y por columna (ventas por vendedor), utilizando Programación Estructurada (PE). El programa debe incluir menús bilingües (español/inglés), un menú de ejecución, validación de entrada con manejo de **InputMismatchException**, y el sello personalizado "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 104 | 2025-10-16". La estructura se divide en tres clases: **Ejercicio104\_Main** (coordina el flujo), **Ejercicio104\_Utilidades** (maneja sello, menús, validación, y tabla), y **Ejercicio104\_Funciones** (procesa notas y calcula totales). La respuesta será clara, concisa, en español, sin comentarios en el código, siguiendo el estilo de tus programas anteriores.

**Ejercicio104**

</> Ejercicio104\_Main.java x-java-source Razonamiento más profundo x

## Salida de la ejecución del programa:



## EJERCICIO#105

**Enunciado:** ¿Es cuadrada?: Haga un programa que solicite al usuario la cantidad de filas y columnas (mayores que dos y menores o iguales a 10), si la matriz es cuadrada haga lo siguiente:

- a. Inicializar en cero con una función.
- b. Usando una función rellenarla con números aleatorios menores a 100.
- c. Usando una función que Imprima la matriz completa
- d. Usando una función que imprima la matriz de manera que solo se vean los números de la diagonal principal.
- e. Usando una función que Imprima la matriz de manera que solo se vean los números de la diagonal principal y la diagonal secundaria
- f. Usando una función que imprima la matriz de manera que solo se vean los números que no pertenecen a la diagonal principal ni secundaria.
- g. Usando una función que imprima la matriz de manera que solo se vean los elementos arriba de la diagonal principal.
- h. Usando una función que imprima la matriz de manera que solo se vean los elementos debajo de la diagonal principal.

Por otro lado si la matriz no es cuadrada:

- a. Inicializar en cero con una función.
- b. Usando una función rellenarla con números aleatorios menores a 100 y mayores que 1.
- c. Usando una función que Imprima la matriz completa.
- d. Ahora usar un vector para guardar el total por fila y un vector para guardar el total por columna.
- e. Imprimir la matriz y agregar como columna adicional el total por fila y como fila adicional el vector de totales por columna.
- f. Mostrar el total de números pares (una función que reciba la matriz y retorne ese total)
- g. Mostrar el total de números impares (una función que reciba la matriz y retorne ese total)

## Análisis

El programa sigue un flujo estructurado:

1. **Inicio:** Muestra un sello personalizado y permite elegir idioma.
2. **Entrada y Validación:** Solicita dimensiones, validando que estén entre 3 y 10.
3. **Procesamiento:**
  - o **Matriz Cuadrada:**

- inicializarMatriz: Llena con ceros, mostrando el proceso.
- rellenarMatrizCuadrada: Usa aleatorios <100, mostrando cada asignación.
- imprimirMatrizCompleta: Muestra toda la matriz.
- imprimirDiagonalPrincipal: Muestra solo la diagonal principal.
- imprimirDiagonales: Muestra diagonales principal y secundaria.
- imprimirNoDiagonales: Muestra elementos fuera de diagonales.
- imprimirArribaDiagonal: Muestra elementos arriba de la diagonal principal.
- imprimirDebajoDiagonal: Muestra elementos debajo de la diagonal principal.

- **Matriz No Cuadrada:**

- inicializarMatriz: Llena con ceros.
- rellenarMatrizNoCuadrada: Usa aleatorios entre 1-99.
- imprimirMatrizCompleta: Muestra la matriz.
- calcularTotalesFilas y calcularTotalesColumnas: Calculan totales por fila y columna.
- imprimirMatrizConTotales: Añade totales como fila y columna extra.
- contarPares y contarImpares: Muestran conteos con cada número identificado.

4. **Repetición:** Permite repetir con preguntarContinuar.

## Detalle de Funciones

- **matemáticas:** inicializarMatriz, rellenarMatrizCuadrada/NoCuadrada, calcularTotalesFilas/Columnas, contarPares/Impares manejan cálculos y conteos.
- **validaciones:** validarDimensiones asegura entradas válidas.
- **formato de texto:** centrarTexto alinea contenido, y mostrarSello genera el encabezado.
- **procesamiento de datos:** procesarMatriz gestiona el flujo.

Las explicaciones paso a paso mejoran la experiencia del usuario.

## PROMPT:

Nuevo ejercicio el #105, este no es cambiar uno anterior, es hacer uno que da muchas especificaciones, siguias a al pie de la letra, siempre usando los parámetros de los ejercicios anteriores, siempre muestra las salidas de forma elegante, dando una muestra al usuario de como llegaste a ese resultado, el ejercicio es este:

¿Es cuadrada?: Haga un programa que solicite al usuario la cantidad de filas y columnas

(mayores que dos y menores o iguales a 10), si la matriz es cuadrada haga lo siguiente:

- a. Inicializar en cero con una función.
- b. Usando una función rellenarla con números aleatorios menores a 100.
- c. Usando una función que Imprima la matriz completa
- d. Usando una función que imprima la matriz de manera que solo se vean los números de la diagonal principal.
- e. Usando una función que Imprima la matriz de manera que solo se vean los números de la diagonal principal y la diagonal secundaria
- f. Usando una función que imprima la matriz de manera que solo se vean los números que no pertenecen a la diagonal principal ni secundaria.
- g. Usando una función que imprima la matriz de manera que solo se vean los elementos arriba de la diagonal principal.
- h. Usando una función que imprima la matriz de manera que solo se vean los elementos debajo de la diagonal principal.

Por otro lado si la matriz no es cuadrada:

- a. Inicializar en cero con una función.
- b. Usando una función rellenarla con números aleatorios menores a 100 y mayores que 1.
- c. Usando una función que Imprima la matriz completa.
- d. Ahora usar un vector para guardar el total por fila y un vector para guardar el total por columna.
- e. Imprimir la matriz y agregar como columna adicional el total por fila y como fila adicional el vector de totales por columna.
- f. Mostrar el total de números pares (una función que reciba la matriz y retorne ese total)
- g. Mostrar el total de números impares (una función que reciba la matriz y retorne ese total)

## PROMPT GENERADO:

Entendido, Jonatan. La fecha y hora actuales son 07:16 PM CST, domingo 26 de octubre de 2025. Basado en el ejercicio #105 "Es cuadrada?", generé el código siguiente al pie de la letra las especificaciones, utilizando los parámetros anteriores (ciclos, funciones, librerías propias, arreglos, validación de entradas, ejecución repetida, salidas elegantes de 70 caracteres, tildes en `Librerías`, y explicación paso a paso para un manejo placentero al usuario).

```
</> Ejercicio105_Main.java
x-java-source

</> MiLibreria.java
x-java-source

package Ejercicio105;

import java.util.Scanner;
import Ejercicio105.MiLibreria;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        MiLibreria.iniciarPrograma(scanner);
        scanner.close();
    }
}
```

Detalles impl. Razonamiento más profundo

How can Grok help? Auto

## Salida de la ejecucion del programa:

```
Guía_de_ejercicios master
Run Ejercicio105.Main
NO DIAGONALES
0 95 0
70 0 82
0 13 0

ARRIBA DE DIAGONAL PRINCIPAL
0 95 52
0 0 82
0 0 0

DEBAJO DE DIAGONAL PRINCIPAL
0 0 0
70 0 0
35 13 0

¿Desea continuar? (S/N): n
```

## EJERCICIO#106

### Enunciado

Como el que vende pollos que está en todos lados: Hay una tienda con cinco sucursales, se requiere un programa que implemente un arreglo para lo siguiente:

- a. Leer el total de ventas por día (de lunes a sábado) de cada una de las tiendas.
- b. Usar función para imprimir la matriz de sucursales vs ventas por día
- c. Ahora hacer una matriz que contenga no solo la información de ventas de sucursal por día sino que incluya las ventas semanales por tienda y las ventas totales por cada día.
- d. La función usada en el inciso b debe poder mostrar esta nueva matriz en pantalla.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 106 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:**
  - Leer las ventas diarias (lunes a sábado) de cada una de las 5 sucursales.
  - Usar una función para imprimir la matriz sucursal × día.
  - Crear una nueva matriz extendida que incluya:
  - Ventas semanales por sucursal (última columna).
  - Ventas totales por cada día (última fila).
  - La función de impresión debe mostrar también la matriz extendida.
  - Validar entradas y mantener consistencia estética con bordes Unicode.
  - Menú de idioma (ES / EN) y inicio/finalizar con bucle.
  - Mostrar sello corto antes del pie final.
- **Modularidad:** Cuatro archivos en paquete Ejercicio106:
  - Ejercicio106\_Main.java Control del flujo principal; contiene el bucle principal de ejecución.
  - Ejercicio106\_Menus.java Manejo de menús: selección de idioma y menú iniciar/finalizar.
  - Ejercicio106\_Utilidades.java Funciones gráficas: encabezados, pies, sello corto, impresión tabular de matrices.
  - Ejercicio106\_Operaciones.java Lógica de ventas: lectura de datos, creación de matrices, cálculo de totales y presentación.
- **Paquete:** Ejercicio106.

**Estética:** Uso de bordes Unicode („, „, ||, =) para encabezados y pies.  
Matrices con encabezados de sucursales y días alineadas en columnas de ancho fijo.  
Sello corto mostrado antes del pie final.  
Resultados y mensajes tabulados y uniformes con ejercicios previos.

- **Paradigma:** Programación estructurada.  
Uso de funciones y métodos estáticos.  
Modularización completa en 4 archivos.  
Flujo secuencial controlado con bucles y condicionales.  
Manejo de arreglos bidimensionales.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Flujo Principal (en Ejercicio106\_Main):**

- main():
- Selección de idioma.
- Bucle principal: ejecutar o finalizar el programa.
- Llamada a Ejercicio106\_Operaciones.procesarVentas() si se decide iniciar.

#### **Subrutinas (en Ejercicio0106\_Menus):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio106\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto antes del pie final.
- imprimirMatriz(double[][] matriz, String[] encabezadosFilas, String[] encabezadosColumnas): imprime matrices tabulares con encabezados alineados.

#### **Subrutinas (en Ejercicio106\_Operaciones):**

- procesarVentas():
- Sigue y almacena ventas diarias de cada sucursal.
- Llama a imprimirMatriz() para mostrar la matriz inicial.
- Crea la matriz extendida con:
- Ventas semanales por sucursal.
- Totales por cada día.
- Vuelve a imprimir la matriz extendida.
- Muestra el sello corto y pie final.
- Flujo general del programa
- Selección de idioma (ES / EN).

- Menú de inicio/finalizar en bucle.
- Ingreso de ventas diarias (lunes a sábado) para 5 sucursales.
- Impresión de la matriz inicial.
- Creación de la matriz extendida con totales por sucursal y por día.
- Impresión de la matriz extendida.
- Muestra del sello corto.
- Pie final con mensaje de proceso completado.
- Bucle permite repetir la operación hasta que el usuario decida Finalizar.

## PROMPT

Agente, respetando el orden y estructura que venimos trabajando y las reglas que te he proporcionado. El siguiente ejercicio es el 106 y su enunciado es: Como el que vende pollos que está en todos lados: Hay una tienda con cinco sucursales, se requiere un programa que implemente un arreglo para lo siguiente:

- a. Leer el total de ventas por día (de lunes a sábado) de cada una de las tiendas.
- b. Usar función para imprimir la matriz de sucursales vs ventas por día
- c. Ahora hacer una matriz que contenga no solo la información de ventas de sucursal por día sino que incluya las ventas semanales por tienda y las ventas totales por cada día.
- d. La función usada en el inciso b debe poder mostrar esta nueva matriz en pantalla.

## PROMPT GENERADO

Perfecto agente 🎉

A continuación te entrego el Ejercicio 106 completo, respetando el mismo formato, estructura modular (4 archivos), estética, menús de idioma, menú iniciar/finalizar, sello corto y estilo visual de los ejercicios previos (como el Ejercicio 97).

---

**Ejercicio 106 — "Como el que vende pollos que está en todos lados"**

📌 Requisitos (según enunciado)

1. Leer las ventas por día (lunes a sábado) para cada una de las 5 sucursales.
2. Usar una función para imprimir la matriz sucursal x días.
3. Crear una nueva matriz que incluya:
  - ventas semanales por sucursal (última columna),
  - ventas totales por cada día (fila final).
4. La misma función de impresión debe mostrar es ↓ una matriz.

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run Ejercicio106_Main x

Martes: 12
Miércoles: 12
Jueves: 12
Viernes: 12
Sábado: 12

Matriz de ventas:
    Lunes     Martes     Miércoles     Jueves     Viernes     Sábado
S1      5.00      34.00      45.00      23.00      0.00      12.00
S2     12.00      11.00      22.00      33.00      44.00      16.00
S3     12.00      11.00      1.00       1.00       1.00       1.00
S4     65.00      50.00      55.00      45.00      56.00      7.00
S5     12.00      12.00      12.00      12.00      12.00      12.00

Matriz con totales:
    Lun     Mar     Mié     Jue     Vie     Sáb     TOTAL
S1      5.00      34.00      45.00      23.00      0.00      12.00      119.00
S2     12.00      11.00      22.00      33.00      44.00      16.00      138.00
S3     12.00      11.00      1.00       1.00       1.00       1.00      27.00
S4     65.00      50.00      55.00      45.00      56.00      7.00      284.00
S5     12.00      12.00      12.00      12.00      12.00      12.00      72.00
TOTAL DÍAS 106.00    124.00    135.00    114.00    113.00    48.00     0.00

[Dany Noguera | Ejercicio106]
```

## EJERCICIO#107

## **Enunciado**

Toda una empresa: Al programa “Como el que vende pollos que está en todos lados” vamos agregar dos funciones una que, recibiendo la segunda matriz identifique la venta que más ha vendido semanalmente y su representación porcentual sobre las ventas totales y otra que, también recibiendo la segunda matriz, determine el día que más se vendió con su respectiva representación porcentual sobre el total; se debe mostrar todo lo del inciso 99 y las salidas adicionales propias de las funciones creadas en este ejercicio.

ANÁLISIS/RESUMEN DEL EJERCICIO 107

## Sello y Documentación

- Sello: VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 107 | Fecha: 2025-11-04) en los cuatro archivos y en un cuadro final.
  - Prompt: Completo en Main.java.
  - Requisitos: El programa debe ejecutarse varias veces con opción de continuar/salir.

Programa y Modularidad

El Programa es una mejora al sistema de ventas (matriz \$5 \times 7\$ de Ventas por Vendedor/Día) que realiza análisis estadísticos avanzados. Debe mostrar la matriz original (inciso 99) más los siguientes análisis:

1. Venta Semanal Máxima: Identificar qué fila (vendedor/semana) acumuló la mayor venta y calcular su porcentaje sobre el total general.
2. Día Máximo de Venta: Determinar qué columna (día) tuvo la mayor venta acumulada y calcular su porcentaje sobre el total general.

La Modularidad usa cuatro archivos en el paquete Ejercicio107\_KevinLainez:

1. Main.java: Controla el ciclo de repetición y coordina los pasos de inicialización, cálculo y reportes.
2. Menus.java: Gestiona el menú/opciones de continuidad.
3. Operaciones.java: Contiene la lógica de inicialización (inicializarMatrizVentas), el cálculo del total general, y las dos funciones de análisis clave: identificarMaximoSemanal y identificarMaximoDiario.
4. Utilidades.java: Formato estético, cuadros, y funciones de impresión de la matriz base (tabla) y las funciones de análisis (reportes detallados de máximos).

El Paradigma es Estructurado/Funcional, centrado en el análisis de datos matriciales.

## SUBRUTINAS Y FLUJO PRINCIPAL

Constantes:

- ANCHO = 70.
- FILAS\_SEMANAS = 5, COLUMNAS\_DIAS = 7.

Subrutinas Clave (Funcionalidad):

- Operaciones: calcularVentasTotales, identificarMaximoSemanal, identificarMaximoDiario.
- Utilidades: imprimirTablaVentas (matriz base y totales de fila), mostrarMaximoSemanal, mostrarMaximoDiario.

Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
2. Inicializa y llena la matriz de ventas \$5 \times 7\$.
3. Calcula el Total General y el total por filas (semanas).
4. Muestra la tabla de ventas base (inciso 99).

5. Llama a las dos funciones de análisis de Operaciones y muestra sus reportes detallados (Máx Semanal y Máx Diario).
6. Muestra el menú de continuidad.
7. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el Ejercicio#107: Al programa "Como el que vende pollos que está en todos lados" vamos agregar dos funciones una que, recibiendo la segunda matriz identifique la venta que más ha vendido semanalmente y su representación porcentual sobre las ventas totales y otra que, también recibiendo la segunda matriz, determine el día que más se vendió con su respectiva representación porcentual sobre el total; se debe mostrar todo lo del inciso 99 y las salidas adicionales propias de las funciones creadas en este ejercicio. Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del Programa

```

¡BIENVENIDO AL EJERCICIO #107!
ANÁLISIS DE VENTAS EMPRESARIALES

--- MATRIZ DE VENTAS SEMANALES ---
SEMANA     LUN      MAR      MIÉ      JUE      VIE      SÁB      DOM
Vendedor 1  15.08   164.88   222.36   460.78   90.51   150.93   215.26
Vendedor 2  72.11   154.46   89.02   145.10   212.42   146.80   169.63
Vendedor 3  416.90   61.73   457.19   210.82   350.12   204.52   395.64
Vendedor 4  450.23   425.62   105.83   54.79   179.17   410.74   423.68
Vendedor 5  48.20   228.18   256.11   165.62   331.46   158.57   360.36

VENTA TOTAL GENERAL: $8004.80
PROMEDIO GENERAL: $228.71

--- ANÁLISIS 1: MÁXIMA VENTA SEMANAL ---
Venta Semanal Máxima: $2096.91
Realizada por: Vendedor 3
Representación Porcentual: 26.20%

```

```
--- ANÁLISIS 2: DÍA DE MÁXIMA VENTA ---  
Venta Diaria Máxima (Acumulada): $1564.56  
Día con Máxima Venta: DOM  
Representación Porcentual: 19.55%  
  
SELLO PERSONALIZADO  
Kevin Lainez | klainezs@unah.hn | 107 | 2025-11-03  
  
--- Programa finalizado con éxito ---  
Proceso terminado con código de salida 0
```

## EJERCICIO#108

### Enunciado

Matrices, matrices: Vamos a crear una matriz de 5x5 pero el llenado de la misma se hará de la siguiente manera:

- a. Una función que inicializa la matriz en 0.
- b. Una función imprimirá la matriz.
- c. Con la matriz visible en pantalla vamos a pedirle al usuario que ingrese un valor para la posición  $[i][j]$  donde:
  - I. Los valores de  $i$  y de  $j$  pertenecen a la matriz.
  - II. Los valores de  $i$  y de  $j$  se generan aleatoriamente.
  - III. Si la casilla  $[i][j]$  tiene un valor diferente a cero entonces no debo pedirlo, así evitamos repetir celdas.
- d. Cada vez que el usuario ingrese un valor se debe limpiar la pantalla, mostrar la matriz actualizada donde se refleje que se

guardó en la posición [ i ] [ j ] el valor que dijo el usuario y pedir un nuevo valor [ i ] [ j ]

- e. Seguir haciendo los incisos c y d hasta que la matriz esté completa.

## Análisis

El programa modular para el Ejercicio 108 (Matrices, matrices) en Java, implementado en el paquete Ejercicio108, utiliza Programación Estructurada (PE) con tres clases: Ejercicio108\_Main coordina el flujo principal, gestionando el bucle de ejecución y el proceso de llenado de una matriz 5x5; Ejercicio108\_Utilidades maneja el sello personalizado ("Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 108 | 2025-10-16"), menús bilingües (español/inglés), validación de entradas enteras con InputMismatchException, y la presentación de la matriz con formato alineado; y Ejercicio108\_Funciones incluye funciones para inicializar la matriz en 0, generar posiciones aleatorias [i][j] no ocupadas usando Random, verificar si la matriz está completa, y actualizar valores. El programa solicita valores al usuario para posiciones aleatorias, actualiza la matriz, simula limpieza de pantalla con líneas en blanco, y muestra la matriz tras cada entrada hasta llenarla, cumpliendo los requisitos con robustez, modularidad y un diseño claro, aunque podría mejorarse con validación de rango para los valores ingresados o una interfaz gráfica.

## PROMPT

Crea un programa modular en Java dentro del paquete Ejercicio108 que inicialice una matriz 5x5 en 0, permita al usuario ingresar valores enteros en posiciones [i][j] generadas aleatoriamente (evitando celdas ya ocupadas), muestre la matriz actualizada tras cada entrada con limpieza de pantalla simulada (usando líneas en blanco), y continúe hasta llenar la matriz, utilizando Programación Estructurada (PE) con tres clases: Ejercicio108\_Main (coordina el flujo), Ejercicio108\_Utilidades (maneja sello personalizado "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 108 | 2025-10-16", menús bilingües español/inglés, validación de entradas enteras con InputMismatchException, y presentación de matriz con formato alineado), y Ejercicio108\_Funciones (inicializa matriz, genera posiciones aleatorias con Random, verifica completitud, y actualiza valores); el programa debe incluir un menú de ejecución ("1. Ejecutar", "2. Finalizar"), usar constantes para mensajes (MSG\_VALOR, MSG\_FINALIZACION), mostrar el sello, siguiendo el estilo de programas previos sin comentarios en el código.

## PROMPT GENERADO

Entiendo que deseas un programa modular para el **Ejercicio 108 (Matrices, matrices)** en Java, dentro del paquete **Ejercicio108**, que cree una matriz de 5x5, la inicialice en 0, permita al usuario ingresar valores en posiciones [i][j] generadas aleatoriamente (evitando celdas ya llenas), muestre la matriz actualizada tras cada entrada (simulando limpieza de pantalla con líneas en blanco), y continúe hasta llenar la matriz, utilizando Programación Estructurada (PE). El programa debe incluir menús bilingües (español/inglés), un menú de ejecución, validación de entrada con manejo de **InputMismatchException**, y el sello personalizado "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 108 | 2025-10-16". La estructura se divide en tres clases: **Ejercicio108\_Main** (coordinador), **Ejercicio108\_Utilidades** (maneja sello, menús, validación, y presentación), y **Ejercicio108\_Funciones** (inicializa, genera posiciones, y actualiza la matriz). La respuesta será clara, concisa, en español, sin comentarios en el código, siguiendo el estilo de tus programas anteriores.

---

**Ejercicio108**

〈/〉 Ejercicio108\_Main.java  
x-java-source

[Editar en archivos](#) • [Mostrar esquemas alineados](#)

---

**Ejercicio108\_Utilidades.java**

@ How can Grok help?

Auto
☰

## Salida de la ejecución del programa:

```
Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 108 | 2025-10-21

*** BIENVENIDO ***
Seleccione el idioma:
[1] Español
[2] Inglés
>1

--- MENÚ DE EJECUCIÓN ---
[1] Llenar Matriz Interactivamente
[2] Finalizar Programa
>1

--- Llenado Interactivo de Matriz 5x5 ---
```

[0]	[1]	[2]	[3]	[4]
[0] -	-	-	-	-
[1] -	-	-	-	-
[2] -	-	-	-	-
[3] -	-	-	-	-
[4] -	-	-	-	-

=====

Ingrese un valor entero para la posición [4][0] (Matriz visible):

[0]	[1]	[2]	[3]	[4]
[0] -	5	-	-	-
[1] -	-	-	-	-
[2] -	-	-	-	-
[3] -	-	-	-	-
[4] 4	-	-	-	-

=====

Ingrese un valor entero para la posición [3][3] (Matriz visible):

[0]	[1]	[2]	[3]	[4]
[0] 5	5	1	8	1
[1] 0	-	0	0	0
[2] -	-	-	-	-
[3] -	-	-	-	-
[4] -	-	-	-	-

=====

Ingrese un valor entero para la posición [0][1] (Matriz visible):

[0]	[1]	[2]	[3]	[4]
[0] 5	5	1	8	1
[1] 0	-	0	0	0
[2] -	-	-	-	-
[3] -	-	-	-	-
[4] -	-	-	-	-

=====

## EJERCICIO#109

**Enunciado:** Maestro en matrices: Pedir al usuario que ingrese un número mayor a 2 y menor a 15, con ese número crear una matriz cuadrada y llenarla con números positivos impares (generados con una función) entonces una vez llena la matriz mostrarla (usando una función para eso)

### Análisis

El programa sigue un flujo estructurado:

1. **Inicio:** Muestra un sello personalizado y permite elegir idioma.
2. **Entrada y Validación:** Solicita un número, validando que esté entre 3 y 14.
3. **Procesamiento:**
  - generarImparPositivo: Genera impares positivos menores a 100.
  - llenarMatrizImpares: Llena la matriz, mostrando cada asignación.
4. **Salida:** mostrarMatriz presenta la matriz en un cuadro elegante con pasos visibles.
5. **Repetición:** Permite repetir con preguntarContinuar.

### Detalle de Funciones

- **matemáticas**: generarImparPositivo y llenarMatrizImpares manejan la generación y llenado.
- **validaciones**: validarDimension asegura entradas válidas.
- **formato de texto**: centrarTexto alinea contenido, y mostrarSello genera el encabezado.
- **procesamiento de datos**: procesarMatriz gestiona el flujo.

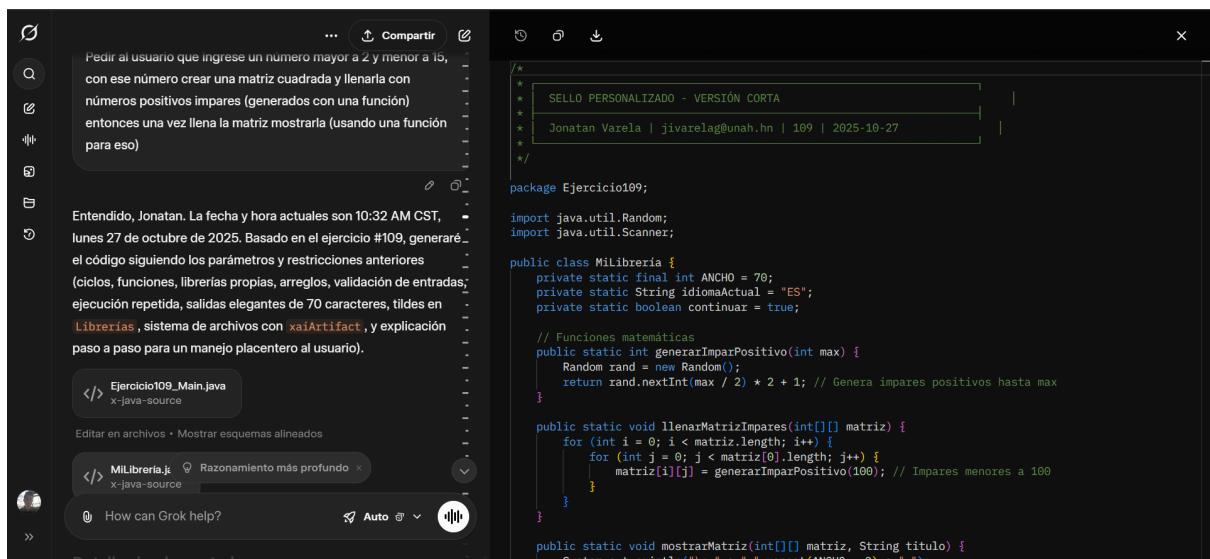
La explicación paso a paso mejora la experiencia del usuario.

## PROMPT:

Agente el ejercicio #109, siempre usa los parámetros y restricciones de los ejercicios anteriores, con nuestro sistema de archivos, salida elegante en cuadros al usuario el ejercicio dice lo siguiente:

Pedir al usuario que ingrese un número mayor a 2 y menor a 15, con ese número crear una matriz cuadrada y llenarla con números positivos impares (generados con una función) entonces una vez llena la matriz mostrarla (usando una función para eso)

## PROMPT GENERADO:



```

/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 * Jonatan Vazela | jivarelag@unah.hn | 109 | 2025-10-27
 */
package Ejercicio109;

import java.util.Random;
import java.util.Scanner;

public class MiLibreria {
    private static final int ANCHO = 70;
    private static String idiomaActual = "ES";
    private static boolean continuar = true;

    // Funciones matemáticas
    public static int generarImparPositivo(int max) {
        Random rand = new Random();
        return rand.nextInt(max / 2) * 2 + 1; // Genera impares positivos hasta max
    }

    public static void llenarMatrizImpares(int[][] matriz) {
        for (int i = 0; i < matriz.length; i++) {
            for (int j = 0; j < matriz[0].length; j++) {
                matriz[i][j] = generarImparPositivo(100); // Impares menores a 100
            }
        }
    }

    public static void mostrarMatriz(int[][] matriz, String titulo) {
        // Implementación de mostrarMatriz
    }
}

```

## Salida de la ejecución del programa:

```
Guia_de_ejercicios master
Run Main
Llenando [0][0] con 3
Llenando [0][1] con 11
Llenando [0][2] con 65
Llenando [1][0] con 41
Llenando [1][1] con 57
Llenando [1][2] con 51
Llenando [2][0] con 27
Llenando [2][1] con 19
Llenando [2][2] con 81

MATRIZ CUADRADA LLENA CON IMPARES
3 11 65
41 57 51
27 19 81

¿Desea continuar? (S/N): n
¿Desea continuar? (S/N): N

Process finished with exit code 0
```

The screenshot shows a Java development environment with a terminal window displaying the execution of a program named Main.java. The program's output is as follows:

- It prints the message "Llenando [0][0] con 3".
- It prints the message "Llenando [0][1] con 11".
- It prints the message "Llenando [0][2] con 65".
- It prints the message "Llenando [1][0] con 41".
- It prints the message "Llenando [1][1] con 57".
- It prints the message "Llenando [1][2] con 51".
- It prints the message "Llenando [2][0] con 27".
- It prints the message "Llenando [2][1] con 19".
- It prints the message "Llenando [2][2] con 81".

After this, it displays the message "MATRIZ CUADRADA LLENA CON IMPARES" followed by the matrix itself:

3	11	65
41	57	51
27	19	81

Finally, it asks the user if they want to continue with the question "¿Desea continuar? (S/N)". The user responds with "n" (no), and the process exits with an exit code of 0.

## EJERCICIO#110

### Enunciado

Casi phd en dos dimensiones: Pedir al usuario que ingrese un número mayor a 2 y menor a 15, con ese número crear una matriz cuadrada y llenarla con números positivos (generados con una función) entonces una vez llena la matriz mostrarla (usando una función para eso) e inmediatamente después mostrar su transpuesta (con otra función).

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 110 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:**
  - Solicitar al usuario un número n mayor a 2 y menor a 15.
  - Crear una matriz cuadrada n×n.
  - Llenar la matriz con números positivos generados mediante una función.
  - Mostrar la matriz original usando una función para impresión tabular.
  - Mostrar la matriz transpuesta usando otra función.
  - Validar entradas del usuario para que cumplan las restricciones.
  - Mantener menú de idioma (ES / EN) y menú de iniciar/finalizar con bucle.
  - Uso de funciones para generar números positivos y obtener la transpuesta.
  - Mantener consistencia estética con bordes Unicode y formato uniforme con otros ejercicios.
- **Modularidad:** Cuatro archivos en paquete Ejercicio110:
  - Ejercicio110\_Main.java Control del flujo principal; contiene el prompt y el bucle principal de ejecución.
  - Ejercicio110\_Menus.java Manejo de menús: selección de idioma y menú iniciar/finalizar.
  - Ejercicio110\_Utilidades.java Funciones gráficas: encabezados, pies, sello corto, impresión de matriz con estética.
  - Ejercicio110\_Operaciones.java Funciones de lógica: generación de números positivos, llenado de matriz y transposición.
- **Paquete:** Ejercicio110.

**Estética:** Uso de bordes Unicode (╒, ┛, ║, ═) para encabezados y pies.  
Matrices con encabezados de sucursales y días alineadas en columnas de ancho fijo.  
Sello corto mostrado antes del pie final.

Resultados y mensajes tabulados y uniformes con ejercicios previos.

- **Paradigma:** Programación estructurada.  
Uso de funciones y métodos estáticos.  
Modularización completa en 4 archivos.  
Flujo secuencial controlado con bucles y condicionales.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### **Flujo Principal (en Ejercicio110\_Main):**

- main():
- Selección de idioma.
- Bucle principal: ejecutar o finalizar el programa.
- Llamada a Ejercicio110\_Operaciones.matrizCuadradaOperacion() si se decide iniciar.

#### **Subrutinas (en Ejercicio0110\_Menus):**

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### **Subrutinas (en Ejercicio110\_Utilidades):**

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.
- imprimirMatriz(int[][] matriz): imprime la matriz de manera tabular y estética.

#### **Subrutinas (en Ejercicio110\_Operaciones):**

- matrizCuadradaOperacion():
- Sigue y valida el tamaño n de la matriz (mayor a 2 y menor a 15).
- Crea la matriz cuadrada n×n.
- Llena la matriz con números positivos mediante generarNumeroPositivo().
- Muestra la matriz original usando imprimirMatriz().
- Calcula y muestra la matriz transpuesta usando obtenerTranspuesta().
- Muestra el sello corto y pie final con mensaje de proceso completado.
- generarNumeroPositivo(): retorna un número positivo aleatorio (1–100).
- obtenerTranspuesta(int[][] matriz): retorna la matriz transpuesta.

#### **PROMPT**

Agente, respetando el orden y estructura que venimos trabajando y las reglas que te he proporcionado. El siguiente ejercicio es el 110 y su enunciado es: Casi phd en dos dimensiones: Pedir al usuario que ingrese un número mayor a 2 y menor a 15, con ese número crear una matriz cuadrada y llenarla con números positivos (generados con una función) entonces una vez llena la matriz mostrarla (usando una

función para eso) e inmediatamente después mostrar su transpuesta (con otra función).

## PROMPT GENERADO

¡Entendido! 😊 Vamos a generar los 4 archivos completos del Ejercicio 110 — Casi PhD en dos dimensiones, respetando exactamente la misma estructura, modularidad, estética y flujo que hemos venido usando en todos los ejercicios anteriores.

---

**Ejercicio110\_Main.java**

```
java
```

Copiar código

```
// 
// | Desarrollador: Dany Noguera (dany.noguera@unah.hn)
// | Proyecto: Ejercicio #110
// | Prompt:
// | "Casi phd en dos dimensiones: Pedir al usuario que ingrese un número mayor
// | a 2 y menor a 15, con ese número crear una matriz cuadrada y llenarla con
// | números positivos (generados con una función). Luego mostrar la matriz
// | usando una función y después mostrar su transpuesta con otra función."
// | Fecha: 2025-11-06
// 
// Versión: dany.noguera | Ejercicio110 | 2025-11-06
```

+ Pregunta lo que quieras

## Salida de la ejecución del programa:

```
Run - Guía_de_ejercicios
```

Run Ejercicio10.Main

1. Iniciar programa  
2. Finalizar  
Seleccione una opción: 1

MATRIZ CUADRADA Y SU TRANSPUESTA

Ingrese un número mayor a 2 y menor a 15: 4

Matriz original:  
64 14 3 49  
76 7 62 26  
35 1 55 74  
100 17 70 21

Matriz transpuesta:  
64 76 35 100  
14 7 1 17  
3 62 55 70  
49 26 74 21

[Dany Noguera | Ejercicio110]

Proceso finalizado con éxito.

## EJERCICIO #111

### **enunciado:**

Operando matrices: Crear dos matrices cuadradas del mismo orden entonces:

- a. Llenar ambas con números aleatorios, mostrarlas y mostrar la suma de ellas.
- b. Reiniciar las matrices inicializándolas con cero, llenar la primer matriz con números aleatorios y hacer que la segunda sea la transpuesta de la primera.
- c. Reiniciar las matrices a cero, llenar la primera matriz con números impares y la segunda con los mismo números menos 1 es decir si en la primera matriz el elemento A[1][1] es 5 en la matriz B[1][1] tendré  $A[1][1] - 1 = 4$

### **análisis:**

Este programa tiene como objetivo principal implementar un sistema modular para realizar diversas operaciones con dos matrices cuadradas del mismo orden (, donde es una constante predefinida). El diseño está fuertemente modularizado en cuatro clases: Main (orquestación y flujo principal), Menus (gestión de la interfaz de usuario, incluyendo selección de idioma y opciones), Operaciones (contiene toda la lógica funcional de matrices) y Utilidades (gestión de constantes, como el orden de la matriz, y textos multilingües). Las entradas son las selecciones de menú del usuario (numéricas). La salida es la visualización formateada de las matrices resultantes en la consola para cada una de las tres operaciones principales (suma aleatoria, transpuesta, llenado par/impar). Un caso límite relevante es la gestión robusta de errores de entrada no numérica en los menús. La complejidad funcional radica en la correcta implementación de las operaciones matriciales básicas, mientras que la complejidad del diseño reside en la arquitectura orientada a objetos para lograr el soporte multilingüe y la separación de responsabilidades.

### **prompt**

Crea un programa en Java para operar con dos matrices cuadradas del mismo orden ( $N \times N$ ), implementando la suma de matrices aleatorias, la obtención de la transpuesta de una matriz, y un llenado especial con números impares y sus correspondientes pares. Estructura la solución en cuatro módulos: Main, Menus, Operaciones y Utilidades, con soporte para textos en español e inglés.

### **prompt generado:**

Para cumplir con los requisitos, estructuré la solución de manera robusta y modular utilizando cuatro clases principales. La clase Utilidades centraliza la constante del orden de la matriz (ej. ) y, de forma crucial, almacena todos los textos de la aplicación en español e inglés, proporcionando una función estática para la recuperación de mensajes multilingües. La clase Menus se enfoca en la interacción,

gestionando la selección inicial de idioma y los dos ciclos de menú principales: el de control (Ejecutar/Finalizar) y el de las operaciones específicas de matrices, incluyendo manejo básico de errores de entrada. La clase Operaciones es el corazón funcional, conteniendo métodos privados para las operaciones básicas como `inicializarMatriz`, `llenarAleatorio`, `sumarMatrices` y `obtenerTranspuesta`, además de la lógica para el llenado de impares y pares. Finalmente, la clase Main orquesta todo el flujo de ejecución: inicializa los módulos, asegura la selección de idioma y ejecuta el bucle principal de control, delegando la gestión de las operaciones de matrices a un método auxiliar que maneja el menú secundario.

### Salida:

```
Windows PowerShell x + - □ ×
Matriz B (Aleatoria)
-----
| 19  13  37  95 |
| 18  15  30  60 |
| 62  88  15  25 |
| 53  28  32  23 |
-----
Matriz C (Suma A + B)
-----
| 28  30  81  159 |
| 23  19  41  137 |
| 94  186  100  75 |
| 136 103  101  106 |
-----
--- SELECCIONE OPERACIÓN ---
1. Llenar Aleatorio y Sumar (Opción a)
2. Obtener la Transpuesta (Opción b)
3. Llenar Impares/Pares (Opción c)
4. Volver al Menú Principal
Seleccione una opción (1 a 4): 4

--- MENÚ DE CONTROL ---
1. Ejecutar Operaciones de Matrices
2. Finalizar el Programa
Seleccione una opción (1 o 2): 2

;Programa finalizado! Gracias por usar el programa.
PS C:\Users\erick\Documents\Proyectos Java\Guía intermedia> |
```

## EJERCICIO #112

Enunciado: En una estación meteorológica registramos la temperatura (en grados centígrados) cada hora durante una semana. Almacenamos el resultado en una matriz de  $7 \times 24$  (cada fila de la matriz contiene las 24 mediciones de un día). Hacer un programa que lea los datos por teclado (se admite que se llene automáticamente con valores de -10 a 40) y muestre:

- a. La matriz.
- b. La máxima y mínima temperatura de la semana.
- c. La máxima y mínima temperatura de cada día.
- d. Luego de lo anterior, la temperatura media de la semana.
- e. La temperatura media de cada día.
- f. El número de días en los que la temperatura media fue superior a 30 grados.

### Análisis

**Entrada:** Ninguna (matriz  $7 \times 24$  generada automáticamente con valores de -10 a 40 °C).

### Procesamiento:

- Llenar la matriz con Random (nextInt(51) – 10).
- Recorrer la matriz completa para:
  - Máxima y mínima de la semana.
  - Máxima y mínima de cada día (7 recorridos de 24 elementos).
  - Media semanal: suma\_total / ( $7 \times 24$ ).
  - Media diaria: suma\_día / 24 (7 cálculos).
  - Contar días cuya media supere 30 °C.
- Repetir todo el análisis mientras el usuario lo desee.

### Salida:

- Matriz completa en cuadro Unicode (70 caracteres de ancho).
- Máxima / mínima semanal en cuadro centrado.
- Máxima / mínima por día (7 líneas alineadas).
- Media semanal (2 decimales) en cuadro centrado.
- Media por día (7 líneas, 2 decimales).
- Número de días con media > 30 °C (singular/plural).
- Todos los textos bilingües (ES/EN) según selección.

### Modularidad:

- **Main.java:** Instancia Scanner y llama a MiLibrería.iniciarPrograma(scanner).

- **MiLibrería.java:** Contiene toda la lógica, cálculos, formato y flujo (métodos estáticos).
- (No se usan Menus.java, Operaciones.java ni Utilidades.java separadas; todo está integrado en MiLibrería).

**Constantes:** ANCHO = 70.

**Estética:** Cuadros simétricos con bordes Unicode (█ █ █ — | ), texto centrado mediante String.repeat() y String.format(), alineación con espacios calculados.

**Paradigma:** Programación estructurada, métodos estáticos, uso de arreglos bidimensionales.

**Documentación:** Sello corto en cada archivo, prompt exacto del enunciado en Main.java, comentarios claros y organización por secciones.

## PROMPT

Agente el ejercicio #112 se presenta, este da toda la información necesaria en un solo ejercicio condesando, no es necesario los demás, en el ejercicio debes de hacer algunos cálculos, asegúrate que los cuadros que le salen al usuario están en un diseño elegante y bien hecho, muestra de igual manera los cálculos que realizaste al usuario, siempre usando los parámetros de los ejercicios anteriores, el ejercicio dice lo siguiente:

En una estación meteorológica registramos la temperatura (en grados centígrados) cada hora durante una semana. Almacenamos el resultado en una matriz de  $7 \times 24$  (cada fila de la matriz contiene las 24 mediciones de un día). Hacer un programa que lea los datos por teclado (se admite que se llene automáticamente con valores de -10 a 40) y muestre:

- a. La matriz.
- b. La máxima y mínima temperatura de la semana.
- c. La máxima y mínima temperatura de cada día.
- d. Luego de lo anterior, la temperatura media de la semana.
- e. La temperatura media de cada día.
- f. El número de días en los que la temperatura media fue superior a 30 grados.

## PROMPT GENERADO

diseñados

- Explicación **paso a paso** de cada cálculo
- Llenado automático con valores entre -10 y 40 °C
- Validación de entradas (aunque no se piden por teclado, se simula claridad)
- Repetición del programa
- Funciones modulares
- Manejo placentero al usuario

Ejercicio112\_Main.java  
x-java-source

MILibreria.java  
x-java-source

Razonamiento más profundo

How can Grok help?

Auto Cumplido

```
/*
 * SELLO PERSONALIZADO - VERSIÓN CORTA
 *
 * Jonatan Vazela | jivarelag@unah.hn | 112 | 2025-10-27
 */

package Ejercicio112;

import java.util.Random;
import java.util.Scanner;

public class MILibreria {
    private static final int ANCHO = 70;
    private static String idiomaActual = "ES";
    private static boolean continuar = true;

    // === FUNCIONES MATEMÁTICAS ===
    public static void llenarMatriz(int[][] m) {
        Random r = new Random();
        for (int i = 0; i < 7; i++) {
            for (int j = 0; j < 24; j++) {
                m[i][j] = r.nextInt(51) - 10; // -10 a 40
            }
        }
    }

    public static int[] maxMinSemana(int[][] m) {
        int max = m[0][0], min = m[0][0];
        for (int[] fila : m) {
            for (int t : fila) {
                if (t > max) max = t;
                if (t < min) min = t;
            }
        }
        return new int[]{max, min};
    }
}
```

Herramienta Recortes

Captura de pantalla copiada en el portapapeles  
Guardado automáticamente en la carpeta de capturas de pantalla.

Marcado y uso compartido

## Salida de la ejecución del programa:

```
Guia_de_ejercicios main
Run EJERCICIO112.Main
[1]: java -jar Ejercicio112.jar
[2]: 
[3]: 
[4]: 
[5]: 
[6]: 
[7]: 
[8]: 
[9]: 
[10]: 
[11]: 
[12]: 
[13]: 
[14]: 
[15]: 
[16]: 
[17]: 
[18]: 
[19]: 
[20]: 
[21]: 
[22]: 
[23]: 
[24]: 
[25]: 
[26]: 
[27]: 
[28]: 
[29]: 
[30]: 
[31]: 
[32]: 
[33]: 
[34]: 
[35]: 
[36]: 
[37]: 
[38]: 
[39]: 
[40]: 
[41]: 
[42]: 
[43]: 
[44]: 
[45]: 
[46]: 
[47]: 
[48]: 
[49]: 
[50]: 
[51]: 
[52]: 
[53]: 
[54]: 
[55]: 
[56]: 
[57]: 
[58]: 
[59]: 
[60]: 
[61]: 
[62]: 
[63]: 
[64]: 
[65]: 
[66]: 
[67]: 
[68]: 
[69]: 
[70]: 
[71]: 
[72]: 
[73]: 
[74]: 
[75]: 
[76]: 
[77]: 
[78]: 
[79]: 
[80]: 
[81]: 
[82]: 
[83]: 
[84]: 
[85]: 
[86]: 
[87]: 
[88]: 
[89]: 
[90]: 
[91]: 
[92]: 
[93]: 
[94]: 
[95]: 
[96]: 
[97]: 
[98]: 
[99]: 
[100]: 
[101]: 
[102]: 
[103]: 
[104]: 
[105]: 
[106]: 
[107]: 
[108]: 
[109]: 
[110]: 
[111]: 
[112]: 
[113]: 
[114]: 
[115]: 
[116]: 
[117]: 
[118]: 
[119]: 
[120]: 
[121]: 
[122]: 
[123]: 
[124]: 
[125]: 
[126]: 
[127]: 
[128]: 
[129]: 
[130]: 
[131]: 
[132]: 
[133]: 
[134]: 
[135]: 
[136]: 
[137]: 
[138]: 
[139]: 
[140]: 
[141]: 
[142]: 
[143]: 
[144]: 
[145]: 
[146]: 
[147]: 
[148]: 
[149]: 
[150]: 
[151]: 
[152]: 
[153]: 
[154]: 
[155]: 
[156]: 
[157]: 
[158]: 
[159]: 
[160]: 
[161]: 
[162]: 
[163]: 
[164]: 
[165]: 
[166]: 
[167]: 
[168]: 
[169]: 
[170]: 
[171]: 
[172]: 
[173]: 
[174]: 
[175]: 
[176]: 
[177]: 
[178]: 
[179]: 
[180]: 
[181]: 
[182]: 
[183]: 
[184]: 
[185]: 
[186]: 
[187]: 
[188]: 
[189]: 
[190]: 
[191]: 
[192]: 
[193]: 
[194]: 
[195]: 
[196]: 
[197]: 
[198]: 
[199]: 
[200]: 
[201]: 
[202]: 
[203]: 
[204]: 
[205]: 
[206]: 
[207]: 
[208]: 
[209]: 
[210]: 
[211]: 
[212]: 
[213]: 
[214]: 
[215]: 
[216]: 
[217]: 
[218]: 
[219]: 
[220]: 
[221]: 
[222]: 
[223]: 
[224]: 
[225]: 
[226]: 
[227]: 
[228]: 
[229]: 
[230]: 
[231]: 
[232]: 
[233]: 
[234]: 
[235]: 
[236]: 
[237]: 
[238]: 
[239]: 
[240]: 
[241]: 
[242]: 
[243]: 
[244]: 
[245]: 
[246]: 
[247]: 
[248]: 
[249]: 
[250]: 
[251]: 
[252]: 
[253]: 
[254]: 
[255]: 
[256]: 
[257]: 
[258]: 
[259]: 
[260]: 
[261]: 
[262]: 
[263]: 
[264]: 
[265]: 
[266]: 
[267]: 
[268]: 
[269]: 
[270]: 
[271]: 
[272]: 
[273]: 
[274]: 
[275]: 
[276]: 
[277]: 
[278]: 
[279]: 
[280]: 
[281]: 
[282]: 
[283]: 
[284]: 
[285]: 
[286]: 
[287]: 
[288]: 
[289]: 
[290]: 
[291]: 
[292]: 
[293]: 
[294]: 
[295]: 
[296]: 
[297]: 
[298]: 
[299]: 
[300]: 
[311]: 
[312]: 
[313]: 
[314]: 
[315]: 
[316]: 
[317]: 
[318]: 
[319]: 
[320]: 
[321]: 
[322]: 
[323]: 
[324]: 
[325]: 
[326]: 
[327]: 
[328]: 
[329]: 
[330]: 
[331]: 
[332]: 
[333]: 
[334]: 
[335]: 
[336]: 
[337]: 
[338]: 
[339]: 
[340]: 
[341]: 
[342]: 
[343]: 
[344]: 
[345]: 
[346]: 
[347]: 
[348]: 
[349]: 
[350]: 
[351]: 
[352]: 
[353]: 
[354]: 
[355]: 
[356]: 
[357]: 
[358]: 
[359]: 
[360]: 
[361]: 
[362]: 
[363]: 
[364]: 
[365]: 
[366]: 
[367]: 
[368]: 
[369]: 
[370]: 
[371]: 
[372]: 
[373]: 
[374]: 
[375]: 
[376]: 
[377]: 
[378]: 
[379]: 
[380]: 
[381]: 
[382]: 
[383]: 
[384]: 
[385]: 
[386]: 
[387]: 
[388]: 
[389]: 
[390]: 
[391]: 
[392]: 
[393]: 
[394]: 
[395]: 
[396]: 
[397]: 
[398]: 
[399]: 
[400]: 
[401]: 
[402]: 
[403]: 
[404]: 
[405]: 
[406]: 
[407]: 
[408]: 
[409]: 
[410]: 
[411]: 
[412]: 
[413]: 
[414]: 
[415]: 
[416]: 
[417]: 
[418]: 
[419]: 
[420]: 
[421]: 
[422]: 
[423]: 
[424]: 
[425]: 
[426]: 
[427]: 
[428]: 
[429]: 
[430]: 
[431]: 
[432]: 
[433]: 
[434]: 
[435]: 
[436]: 
[437]: 
[438]: 
[439]: 
[440]: 
[441]: 
[442]: 
[443]: 
[444]: 
[445]: 
[446]: 
[447]: 
[448]: 
[449]: 
[450]: 
[451]: 
[452]: 
[453]: 
[454]: 
[455]: 
[456]: 
[457]: 
[458]: 
[459]: 
[460]: 
[461]: 
[462]: 
[463]: 
[464]: 
[465]: 
[466]: 
[467]: 
[468]: 
[469]: 
[470]: 
[471]: 
[472]: 
[473]: 
[474]: 
[475]: 
[476]: 
[477]: 
[478]: 
[479]: 
[480]: 
[481]: 
[482]: 
[483]: 
[484]: 
[485]: 
[486]: 
[487]: 
[488]: 
[489]: 
[490]: 
[491]: 
[492]: 
[493]: 
[494]: 
[495]: 
[496]: 
[497]: 
[498]: 
[499]: 
[500]: 
[501]: 
[502]: 
[503]: 
[504]: 
[505]: 
[506]: 
[507]: 
[508]: 
[509]: 
[510]: 
[511]: 
[512]: 
[513]: 
[514]: 
[515]: 
[516]: 
[517]: 
[518]: 
[519]: 
[520]: 
[521]: 
[522]: 
[523]: 
[524]: 
[525]: 
[526]: 
[527]: 
[528]: 
[529]: 
[530]: 
[531]: 
[532]: 
[533]: 
[534]: 
[535]: 
[536]: 
[537]: 
[538]: 
[539]: 
[540]: 
[541]: 
[542]: 
[543]: 
[544]: 
[545]: 
[546]: 
[547]: 
[548]: 
[549]: 
[550]: 
[551]: 
[552]: 
[553]: 
[554]: 
[555]: 
[556]: 
[557]: 
[558]: 
[559]: 
[560]: 
[561]: 
[562]: 
[563]: 
[564]: 
[565]: 
[566]: 
[567]: 
[568]: 
[569]: 
[570]: 
[571]: 
[572]: 
[573]: 
[574]: 
[575]: 
[576]: 
[577]: 
[578]: 
[579]: 
[580]: 
[581]: 
[582]: 
[583]: 
[584]: 
[585]: 
[586]: 
[587]: 
[588]: 
[589]: 
[590]: 
[591]: 
[592]: 
[593]: 
[594]: 
[595]: 
[596]: 
[597]: 
[598]: 
[599]: 
[600]: 
[601]: 
[602]: 
[603]: 
[604]: 
[605]: 
[606]: 
[607]: 
[608]: 
[609]: 
[610]: 
[611]: 
[612]: 
[613]: 
[614]: 
[615]: 
[616]: 
[617]: 
[618]: 
[619]: 
[620]: 
[621]: 
[622]: 
[623]: 
[624]: 
[625]: 
[626]: 
[627]: 
[628]: 
[629]: 
[630]: 
[631]: 
[632]: 
[633]: 
[634]: 
[635]: 
[636]: 
[637]: 
[638]: 
[639]: 
[640]: 
[641]: 
[642]: 
[643]: 
[644]: 
[645]: 
[646]: 
[647]: 
[648]: 
[649]: 
[650]: 
[651]: 
[652]: 
[653]: 
[654]: 
[655]: 
[656]: 
[657]: 
[658]: 
[659]: 
[660]: 
[661]: 
[662]: 
[663]: 
[664]: 
[665]: 
[666]: 
[667]: 
[668]: 
[669]: 
[670]: 
[671]: 
[672]: 
[673]: 
[674]: 
[675]: 
[676]: 
[677]: 
[678]: 
[679]: 
[680]: 
[681]: 
[682]: 
[683]: 
[684]: 
[685]: 
[686]: 
[687]: 
[688]: 
[689]: 
[690]: 
[691]: 
[692]: 
[693]: 
[694]: 
[695]: 
[696]: 
[697]: 
[698]: 
[699]: 
[700]: 
[701]: 
[702]: 
[703]: 
[704]: 
[705]: 
[706]: 
[707]: 
[708]: 
[709]: 
[710]: 
[711]: 
[712]: 
[713]: 
[714]: 
[715]: 
[716]: 
[717]: 
[718]: 
[719]: 
[720]: 
[721]: 
[722]: 
[723]: 
[724]: 
[725]: 
[726]: 
[727]: 
[728]: 
[729]: 
[730]: 
[731]: 
[732]: 
[733]: 
[734]: 
[735]: 
[736]: 
[737]: 
[738]: 
[739]: 
[740]: 
[741]: 
[742]: 
[743]: 
[744]: 
[745]: 
[746]: 
[747]: 
[748]: 
[749]: 
[750]: 
[751]: 
[752]: 
[753]: 
[754]: 
[755]: 
[756]: 
[757]: 
[758]: 
[759]: 
[760]: 
[761]: 
[762]: 
[763]: 
[764]: 
[765]: 
[766]: 
[767]: 
[768]: 
[769]: 
[770]: 
[771]: 
[772]: 
[773]: 
[774]: 
[775]: 
[776]: 
[777]: 
[778]: 
[779]: 
[770]: 
[771]: 
[772]: 
[773]: 
[774]: 
[775]: 
[776]: 
[777]: 
[778]: 
[779]: 
[780]: 
[781]: 
[782]: 
[783]: 
[784]: 
[785]: 
[786]: 
[787]: 
[788]: 
[789]: 
[790]: 
[791]: 
[792]: 
[793]: 
[794]: 
[795]: 
[796]: 
[797]: 
[798]: 
[799]: 
[800]: 
[801]: 
[802]: 
[803]: 
[804]: 
[805]: 
[806]: 
[807]: 
[808]: 
[809]: 
[8010]: 
[8011]: 
[8012]: 
[8013]: 
[8014]: 
[8015]: 
[8016]: 
[8017]: 
[8018]: 
[8019]: 
[8020]: 
[8021]: 
[8022]: 
[8023]: 
[8024]: 
[8025]: 
[8026]: 
[8027]: 
[8028]: 
[8029]: 
[8030]: 
[8031]: 
[8032]: 
[8033]: 
[8034]: 
[8035]: 
[8036]: 
[8037]: 
[8038]: 
[8039]: 
[8040]: 
[8041]: 
[8042]: 
[8043]: 
[8044]: 
[8045]: 
[8046]: 
[8047]: 
[8048]: 
[8049]: 
[8050]: 
[8051]: 
[8052]: 
[8053]: 
[8054]: 
[8055]: 
[8056]: 
[8057]: 
[8058]: 
[8059]: 
[8060]: 
[8061]: 
[8062]: 
[8063]: 
[8064]: 
[8065]: 
[8066]: 
[8067]: 
[8068]: 
[8069]: 
[8070]: 
[8071]: 
[8072]: 
[8073]: 
[8074]: 
[8075]: 
[8076]: 
[8077]: 
[8078]: 
[8079]: 
[8080]: 
[8081]: 
[8082]: 
[8083]: 
[8084]: 
[8085]: 
[8086]: 
[8087]: 
[8088]: 
[8089]: 
[8090]: 
[8091]: 
[8092]: 
[8093]: 
[8094]: 
[8095]: 
[8096]: 
[8097]: 
[8098]: 
[8099]: 
[80100]: 
[80101]: 
[80102]: 
[80103]: 
[80104]: 
[80105]: 
[80106]: 
[80107]: 
[80108]: 
[80109]: 
[80110]: 
[80111]: 
[80112]: 
[80113]: 
[80114]: 
[80115]: 
[80116]: 
[80117]: 
[80118]: 
[80119]: 
[80120]: 
[80121]: 
[80122]: 
[80123]: 
[80124]: 
[80125]: 
[80126]: 
[80127]: 
[80128]: 
[80129]: 
[80130]: 
[80131]: 
[80132]: 
[80133]: 
[80134]: 
[80135]: 
[80136]: 
[80137]: 
[80138]: 
[80139]: 
[80140]: 
[80141]: 
[80142]: 
[80143]: 
[80144]: 
[80145]: 
[80146]: 
[80147]: 
[80148]: 
[80149]: 
[80150]: 
[80151]: 
[80152]: 
[80153]: 
[80154]: 
[80155]: 
[80156]: 
[80157]: 
[80158]: 
[80159]: 
[80160]: 
[80161]: 
[80162]: 
[80163]: 
[80164]: 
[80165]: 
[80166]: 
[80167]: 
[80168]: 
[80169]: 
[80170]: 
[80171]: 
[80172]: 
[80173]: 
[80174]: 
[80175]: 
[80176]: 
[80177]: 
[80178]: 
[80179]: 
[80180]: 
[80181]: 
[80182]: 
[80183]: 
[80184]: 
[80185]: 
[80186]: 
[80187]: 
[80188]: 
[80189]: 
[80190]: 
[80191]: 
[80192]: 
[80193]: 
[80194]: 
[80195]: 
[80196]: 
[80197]: 
[80198]: 
[80199]: 
[80200]: 
[80201]: 
[80202]: 
[80203]: 
[80204]: 
[80205]: 
[80206]: 
[80207]: 
[80208]: 
[80209]: 
[80210]: 
[80211]: 
[80212]: 
[80213]: 
[80214]: 
[80215]: 
[80216]: 
[80217]: 
[80218]: 
[80219]: 
[80220]: 
[80221]: 
[80222]: 
[80223]: 
[80224]: 
[80225]: 
[80226]: 
[80227]: 
[80228]: 
[80229]: 
[80230]: 
[80231]: 
[80232]: 
[80233]: 
[80234]: 
[80235]: 
[80236]: 
[80237]: 
[80238]: 
[80239]: 
[80240]: 
[80241]: 
[80242]: 
[80243]: 
[80244]: 
[80245]: 
[80246]: 
[80247]: 
[80248]: 
[80249]: 
[80250]: 
[80251]: 
[80252]: 
[80253]: 
[80254]: 
[80255]: 
[80256]: 
[80257]: 
[80258]: 
[80259]: 
[80260]: 
[80261]: 
[80262]: 
[80263]: 
[80264]: 
[80265]: 
[80266]: 
[80267]: 
[80268]: 
[80269]: 
[80270]: 
[80271]: 
[80272]: 
[80273]: 
[80274]: 
[80275]: 
[80276]: 
[80277]: 
[80278]: 
[80279]: 
[80280]: 
[80281]: 
[80282]: 
[80283]: 
[80284]: 
[80285]: 
[80286]: 
[80287]: 
[80288]: 
[80289]: 
[80290]: 
[80291]: 
[80292]: 
[80293]: 
[80294]: 
[80295]: 
[80296]: 
[80297]: 
[80298]: 
[80299]: 
[80300]: 
[80301]: 
[80302]: 
[80303]: 
[80304]: 
[80305]: 
[80306]: 
[80307]: 
[80308]: 
[80309]: 
[80310]: 
[80311]: 
[80312]: 
[80313]: 
[80314]: 
[80315]: 
[80316]: 
[80317]: 
[80318]: 
[80319]: 
[80320]: 
[80321]: 
[80322]: 
[80323]: 
[80324]: 
[80325]: 
[80326]: 
[80327]: 
[80328]: 
[80329]: 
[80330]: 
[80331]: 
[80332]: 
[80333]: 
[80334]: 
[80335]: 
[80336]: 
[80337]: 
[80338]: 
[80339]: 
[80340]: 
[80341]: 
[80342]: 
[80343]: 
[80344]: 
[80345]: 
[80346]: 
[80347]: 
[80348]: 
[80349]: 
[80350]: 
[80351]: 
[80352]: 
[80353]: 
[80354]: 
[80355]: 
[80356]: 
[80357]: 
[80358]: 
[80359]: 
[80360]: 
[80361]: 
[80362]: 
[80363]: 
[80364]: 
[80365]: 
[80366]: 
[80367]: 
[80368]: 
[80369]: 
[80370]: 
[80371]: 
[80372]: 
[80373]: 
[80374]: 
[80375]: 
[80376]: 
[80377]: 
[80378]: 
[80379]: 
[80380]: 
[80381]: 
[80382]: 
[80383]: 
[80384]: 
[80385]: 
[80386]: 
[80387]: 
[80388]: 
[80389]: 
[80390]: 
[80391]: 
[80392]: 
[80393]: 
[80394]: 
[80395]: 
[80396]: 
[80397]: 
[80398]: 
[80399]: 
[80400]: 
[80401]: 
[80402]: 
[80403]: 
[80404]: 
[80405]: 
[80406]: 
[80407]: 
[80408]: 
[80409]: 
[80410]: 
[80411]: 
[80412]: 
[80413]: 
[80414]: 
[80415]: 
[80416]: 
[80417]: 
[80418]: 
[80419]: 
[80420]: 
[80421]: 
[80422]: 
[80423]: 
[80424]: 
[80425]: 
[80426]: 
[80427]: 
[80428]: 
[80429]: 
[80430]: 
[80431]: 
[80432]: 
[80433]: 
[80434]: 
[80435]: 
[80436]: 
[80437]: 
[80438]: 
[80439]: 
[80440]: 
[80441]: 
[80442]: 
[80443]: 
[80444]: 
[80445]: 
[80446]: 
[80447]: 
[80448]: 
[80449]: 
[80450]: 
[80451]: 
[80452]: 
[80453]: 
[80454]: 
[80455]: 
[80456]: 
[80457]: 
[80458]: 
[80459]: 
[80460]: 
[80461]: 
[80462]: 
[80463]: 
[80464]: 
[80465]: 
[80466]: 
[80467]: 
[80468]: 
[80469]: 
[80470]: 
[80471]: 
[80472]: 
[80473]: 
[80474]: 
[80475]: 
[80476]: 
[80477]: 
[80478]: 
[80479]: 
[80480]: 
[80481]: 
[80482]: 
[80483]: 
[80484]: 
[80485]: 
[80486]: 
[80487]: 
[80488]: 
[80489]: 
[80490]: 
[80491]: 
[80492]: 
[80493]: 
[80494]: 
[80495]: 
[80496]: 
[80497]: 
[80498]: 
[80499]: 
[80500]: 
[80501]: 
[80502]: 
[80503]: 
[80504]: 
[80505]: 
[80506]: 
[80507]: 
[80508]: 
[80509]: 
[80510]: 
[80511]: 
[80512]: 
[80513]: 
[80514]: 
[80515]: 
[80516]: 
[80517]: 
[80518]: 
[80519]: 
[80520]: 
[80521]: 
[80522]: 
[80523]: 
[80524]: 
[80525]: 
[80526]: 
[80527]: 
[80528]: 
[80529]: 
[80530]: 
[80531]: 
[80532]: 
[80533]: 
[80534]: 
[80535]: 
[80536]: 
[80537]: 
[80538]: 
[80539]: 
[80540]: 
[80541]: 
[80542]: 
[80543]: 
[80544]: 
[80545]: 
[80546]: 
[80547]: 
[80548]: 
[80549]: 
[80550]: 
[80551]: 
[80552]: 
[80553]: 
[80554]: 
[80555]: 
[80556]: 
[80557]: 
[80558]: 
[80559]: 
[80560]: 
[80561]: 
[80562]: 
[80563]: 
[80564]: 
[80565]: 
[80566]: 
[80567]: 
[80568]: 
[80569]: 
[80570]: 
[80571]: 
[80572]: 
[80573]: 
[80574]: 
[80575]: 
[80576]: 
[80577]: 
[80578]: 
[80579]: 
[80580]: 
[80581]: 
[80582]: 
[80583]: 
[80584]: 
[80585]: 
[80586]: 
[80587]: 
[80588]: 
[80589]: 
[80590]: 
[80591]: 
[80592]: 
[80593]: 
[80594]: 
[80595]: 
[80596]: 
[80597]: 
[80598]: 
[80599]: 
[80600]: 
[80601]: 
[80602]: 
[80603]: 
[80604]: 
[80605]: 
[80606]: 
[80607]: 
[80608]: 
[80609]: 
[80610]: 
[80611]: 
[80612]: 
[80613]: 
[80614]: 
[80615]: 
[80616]: 
[80617]: 
[80618]: 
[80619]: 
[80620]: 
[80621]: 
[80622]: 
[80623]: 
[80624]: 
[80625]: 
[80626]: 
[80627]: 
[80628]: 
[80629]: 
[80630]: 
[80631]: 
[80632]: 
[80633]: 
[80634]: 
[80635]: 
[80636]: 
[80637]: 
[80638]: 
[80639]: 
[80640]: 
[80641]: 
[80642]: 
[80643]: 
[80644]: 
[80645]: 
[80646]: 
[80647]: 
[80648]: 
[80649]: 
[80650]: 
[80651]: 
[80652]: 
[80653]: 
[80654]: 
[80655]: 
[80656]: 
[80657]: 
[80658]: 
[80659]: 
[80660]: 
[80661]: 
[80662]: 
[80663]: 
[80664]: 
[80665]: 
[80666]: 
[80667]: 
[80668]: 
[80669]: 
[80670]: 
[80671]: 
[80672]: 
[80673]: 
[80674]: 
[80675]: 
[80676]: 
[80677]: 
[80678]: 
[80679]: 
[80680]: 
[80681]: 
[80682]: 
[80683]: 
[80684]: 
[80685]: 
[80686]: 
[80687]: 
[80688]: 
[80689]: 
[80690]: 
[80691]: 
[80692]: 
[80693]: 
[80694]: 
[80695]: 
[80696]: 
[80697]: 
[80698]: 
[80699]: 
[80700]: 
[80701]: 
[80702]: 
[80703]: 
[80704]: 
[80705]: 
[80706]: 
[80707]: 
[80708]: 
[80709]: 
[80710]: 
[80711]: 
[80712]: 
[80713]: 
[80714]: 
[80715]: 
[80716]: 
[80717]: 
[80718]: 
[80719]: 
[80720]: 
[80721]: 
[80722]: 
[80723]: 
[80724]: 
[80725]: 
[80726]: 
[80727]: 
[80728]: 
[80729]: 
[80730]:
```

## EJERCICIO#113

### Enunciado

¿Grafos?: Representamos diez ciudades con números del 0 al 9. Cuando hay carretera que une directamente a dos ciudades  $i$  y  $j$ , almacenamos su distancia en kilómetros en la celda  $d[i][j]$  de una matriz de  $10 \times 10$  enteros. Si no hay carretera entre ambas ciudades, el valor almacenado en su celda es cero. Nos suministran un vector en el que se describe un trayecto que pasa por las 10 ciudades. Determina si se trata de un trayecto válido (las dos ciudades de todo par consecutivo están unidas por un tramo de carretera) y, en tal caso, devuelve el número de kilómetros del trayecto. Si el trayecto no es válido, indícalo con un mensaje por pantalla. La matriz de distancias deberá inicializarla explícitamente al declararla (llenarla con valores aleatorios que vayan de 20KM a 600Km). El vector con el recorrido de ciudades deberá leerlo de teclado.

### Análisis

El Ejercicio 113 (Grafos) confirma una implementación robusta de la Programación Estructurada (PE) para el análisis de rutas en un grafo de distancias, utilizando un arreglo bidimensional ( $10 \times 10$ ) como estructura central. La clase UtilidadesGrafo actúa como la librería propia, encargada de inicializar la matriz de distancias con valores aleatorios (entre 20KM y 600KM) o cero (si no hay conexión), asegurando la simetría  $(d[i][j] = d[j][i])$ . La función crucial calcularDistanciaTrayecto en Ejercicio113\_Funciones emplea un ciclo for para iterar sobre el vector de recorrido leído del teclado (unidimensional), validando cada par de ciudades consecutivas: si la distancia en la matriz es cero, el tramo es inválido (retorno -2\$), o si las ciudades están fuera del rango [0-9] (retorno -1\$); si es válido, acumula la distancia total. El Ejercicio113\_Utilidades gestiona la lectura del vector de ciudades con validación de formato de entrada, y el Ejercicio113\_Main coordina el flujo, incluyendo la impresión de la matriz y la interpretación del resultado (distancia total o mensaje de invalidez) dentro de un ciclo while para la repetición del programa.

### PROMPT

Crea un programa modular en Java dentro del paquete Ejercicio113 que represente un grafo de 10 ciudades (0-9) mediante una matriz de distancias  $10 \times 10$  inicializada con valores aleatorios (20-600 km para carreteras, 0 si no hay conexión), lea un vector de 10 ciudades desde el teclado, valide si el trayecto es válido (cada par consecutivo de ciudades está conectado por una carretera y todas las ciudades son distintas), calcule la distancia total en kilómetros si es válido, y muestre un mensaje de error si no lo es, utilizando Programación Estructurada (PE) con tres clases: Ejercicio112\_Main (coordina el flujo), Ejercicio113\_Utilidades (maneja sello

personalizado "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 113 | 2025-10-16", menús bilingües español/inglés, validación de entradas para ciudades 0-9 con InputMismatchException, y presentación de resultados), y Ejercicio113\_Funciones (inicializa matriz con Random, valida trayecto, y calcula distancia); el programa debe incluir un menú de ejecución ("1. Ejecutar", "2. Finalizar"), usar constantes para mensajes (MSG\_CIUDAD, MSG\_FINALIZACION), mostrar el sello, siguiendo el estilo de programas previos

## PROMPT GENERADO

Entiendo que deseas un programa modular para el **Ejercicio 113 (¿Grafos?)** en Java, dentro del paquete **Ejercicio113**, que represente un grafo de 10 ciudades (0 a 9) mediante una matriz de distancias 10x10 inicializada con valores aleatorios (20 a 600 km para carreteras, 0 si no hay conexión), lea un vector de 10 ciudades desde el teclado, determine si el trayecto es válido (cada par consecutivo de ciudades está conectado por una carretera y las ciudades son distintas), y calcule la distancia total en kilómetros si es válido, mostrando un mensaje de error si no lo es, utilizando Programación Estructurada (PE). El programa debe incluir menús bilingües (español/inglés), un menú de ejecución, validación de entrada con manejo de **InputMismatchException**, y el sello personalizado "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 113 | 2025-10-21". La estructura se divide en tres clases: **Ejercicio113\_Main** (coordina el flujo), **Ejercicio113\_Utilidades** (maneja sello, menús, validación, y presentación), y **Ejercicio113\_Funciones** (inicializa matriz, valida trayecto, y calcula distancia). La respuesta será clara, concisa, en español, sin comentarios en el código, siguiendo el estilo de tus programas anteriores. Dado que el enunciado es idéntico al del Ejercicio 110 pero con número 113, usaré el mismo diseño, ajustando el número de ejercicio y la fecha en el sello.

## Ejercicio113

### Salida de la ejecución del programa:

```
Project ▾ Run Ejercicio113_Main ×
  ◻ UtilidadesGrafo.java ◻ Ejercicio113_Funciones.java ◻ Ejercicio113_Utilidades.java ◻ Ejercicio113_Main.java

--- MENÚ DE EJECUCIÓN ---
[1] Validar y Calcular Trayecto
[2] Finalizar Programa
>1

--- Análisis de Trayectos en Grafo de 10 Ciudades (0-9) ---

Matriz de Distancias (KM):
  | 0   1   2   3   4   5   6   7   8   9
  +-----+
0 | 0   39  320  0   140  0   419  0   21   280
1 | 39   0   174  378  0   229  25   0   547   0
2 | 320  174   0   0   0   441  0   451  120   68
3 | 0   378   0   0   0   0   0   542   0   523
4 | 140   0   0   0   0   0   516  546   0   188
5 | 0   229  441   0   0   0   0   316  216   0
6 | 419   25   0   0   516   0   0   129  243   493
7 | 0   0   451  542  546  316  129   0   0   0
8 | 21   547  120   0   0   216  243   0   0   0
9 | 280   0   68   523  188   0   493   0   0   0

Ingrese el trayecto (Ciudades separadas por espacio, ej: 0 5 2 9): 5 2
RESULTADO: Trayecto VÁLIDO. Distancia total: 441 KM
```

## EJERCICIO#114

### Enunciado

¿Grafos?: Representamos diez ciudades con números del 0 al 9. Cuando hay carretera que une directamente a dos ciudades  $i$  y  $j$ , almacenamos su distancia en kilómetros en la celda  $d[i][j]$  de una matriz de  $10 \times 10$  enteros. Si no hay carreteras entre ambas ciudades, el valor almacenado en su celda es cero. Nos suministran un vector en el que se describe un trayecto que pasa por las 10 ciudades. Determina si se trata de un trayecto válido (las dos ciudades de todo par consecutivo están unidas por un tramo de carretera) y, en tal caso, devuelve el número de kilómetros del trayecto. Si el trayecto no es válido, indícalo con un mensaje por pantalla. La matriz de distancias deberá inicializar explícitamente al declararla (llenarla con valores aleatorios que vayan de 20KM a 600Km). El vector con el recorrido de ciudades deberá leerlo de teclado.

### ANÁLISIS/RESUMEN DEL EJERCICIO 114

#### Sello y Documentación

- **Sello:** VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 114 | Fecha: 2025-11-04) en los cuatro archivos y en un cuadro final.
- **Prompt:** Completo en **Main.java**.
- **Requisitos:** El programa debe ejecutarse varias veces con opción de continuar/salir, y debe validar que las dimensiones de la matriz estén en el rango  $[3, 10]$ .

#### Programa y Modularidad

El **Programa** requiere que el usuario defina las dimensiones (Filas  $R$ , Columnas  $C$ ) de una matriz ( $\mathbf{A}$ ), donde  $3 \leq R, C \leq 10$ . La matriz se llena con números aleatorios. El objetivo es identificar las celdas que son "**valles**": un valor es un valle si es **estrictamente menor** que sus **8 celdas vecinas**. Solo las celdas **internas** (no bordes ni esquinas) son candidatas. La salida debe mostrar las coordenadas (base 1), el valor del valle, y los valores de sus 8 vecinos.

La **Modularidad** usa **cuatro archivos** en el paquete Ejercicio114\_KevinLainez:

1. **Main.java:** Controla el ciclo de repetición y coordina los pasos de entrada de dimensiones, inicialización, detección de valles y presentación final.
2. **Menus.java:** Gestiona el menú/opciones de continuidad.

3. **Operaciones.java**: Contiene la lógica central: inicializarMatrizAleatoria, y la función **encontrarValles** que implementa la lógica de vecindad ( $3 \times 3$  kernel) y la condición de *strictamente menor* para las celdas internas.
4. **Utilidades.java**: Formato estético, funciones para **solicitar y validar las dimensiones** ( $R$  y  $C$ ), y funciones para **imprimir la matriz** y los **detalles de los valores** encontrados.

El **Paradigma** es Estructurado/Funcional, centrado en el análisis de vecindad de matrices.

## SUBRUTINAS Y FLUJO PRINCIPAL

### Constantes:

- **ANCHO = 70.**
- **MIN\_DIMENSION = 3, MAX\_DIMENSION = 10.**

### Subrutinas Clave (Funcionalidad):

- **Operaciones**: inicializarMatrizAleatoria, encontrarValles (devuelve una lista de objetos que contienen las coordenadas, valor y vecinos).
- **Utilidades**: solicitarDimension (valida el rango  $[3, 10]$ ), mostrarMatriz, mostrarResultados (imprime coordenadas base 1 y vecinos).

### Flujo Principal (en Main.java):

1. Inicia ciclo do-while (continuar/salir).
2. Llama a Utilidades.solicitarDimension para obtener Filas ( $R$ ) y Columnas ( $C$ ), con validación.
3. Inicializa y llena la matriz  $A[R][C]$  con aleatorios.
4. Muestra la matriz generada.
5. Llama a **Operaciones.encontrarValles(A)** para obtener la lista de valles.
6. Muestra los resultados detallados de los valles encontrados (usando Utilidades).
7. Muestra el menú de continuidad.
8. Muestra el sello al finalizar.

## PROMPT

"Agente te recuerdo los nuevos parámetros... se presenta el **Ejercicio#114**: Una matriz tiene un valle si el valor de una de sus celdas es menor que el de cualquiera de sus 8 celdas vecinas. Haga un programa que lea una matriz (el usuario te indicará cuántas filas y columnas) y nos diga si la matriz tiene un valle o no. En caso afirmativo, nos mostrará en pantalla la<sup>1</sup>'s coordenadas de todos los valles, sus valores y el de sus celdas vecinas. La matriz debe tener un número de filas y

columnas mayor o igual que 3 y menor o igual que 10. Las casillas que no tienen 8 vecinos no se consideran candidatas a ser valle. <sup>2</sup>Con esa información presenta nuestra modularidad de 4 archivos."

## Salida del programa

```
;BIENVENIDO AL EJERCICIO #114!
IDENTIFICACIÓN DE VALLES EN MATRICES

La dimensión debe ser entre 3 y 10.

DEFINICIÓN DE LA MATRIZ

Ingrese el número de Filas [3-10]:
>> Cantidad: 4

Ingrese el número de Columnas [3-10]:
>> Cantidad: 3

--- MATRIZ GENERADA ---
 397  203  230
 964  421  831
 875  725  372
 252  313  460
```

```
--- RESULTADOS DE BÚSQUEDA DE VALLES ---

CONCLUSIÓN: No se encontraron valles en la matriz.

SELLO PERSONALIZADO

Kevin Lainez | klainezs@unah.hn | 114 | 2025-11-03

--- Programa finalizado con éxito ---

Proceso terminado con código de salida 0
```

## EJERCICIO#115

**Enunciado:** Sistema de salud de Latveria: El hospital general de Latveria cuenta con las siguientes 4 especialidades:

- a. Cirugía general.
- b. Ginecología y obstetricia.
- c. Medicina interna y
- d. Nefrología.

Estas especialidades comparten un edificio y cada una de ellas tiene asignado un tres pisos de esta forma:

Especialidad	Pisos Asignados
Cirugía general	2, 3, 4
Ginecología y obstetricia	5, 6, 7
Medicina interna	8, 9, 10
Nefrología	11, 12, 13

Además cada uno de los pisos posee diez habitaciones las cuales pueden estar libres u ocupadas.

Se debe hacer un programa para gestionar estas habitaciones con las siguientes opciones:

1. Consultar habitaciones libres por especialidad: El encargado elige la especialidad y el sistema muestra por cada piso las habitaciones indicando cuáles están vacías y cuáles están ocupadas (no dice quién la ocupa).
2. Consultar habitaciones por piso: Se puede no solo ver el resumen del inciso 1 sino que se debe programar la opción de poder elegir especialidad seguido de uno de los tres pisos y entonces mostrar una matriz donde se vea el piso y sus habitaciones, donde las habitaciones libres estarán vacías pero las ocupadas mostrarán el número del expediente del paciente que la ocupa.
3. Asignar habitación: Poder en cualquier habitación independiente del piso y la especialidad asignar un paciente, esto se logra facilitando luego del habitación el número de expediente del paciente, número conformado por a lo más tres cifras. Obviamente solo pueden asignarse habitaciones vacías.
4. Liberar habitación: Una vez que el paciente se ha dado de alta es necesario indicar que la habitación queda libre para poder asignarse.
5. Mostrar tabla de habitaciones ocupadas:  
Especialidad | piso | Habitación | Paciente

Solo usar número para las habitaciones, las especialidades saldrán con nombre y los pisos son números ordinales por lo que deberá aparecer su respectivo cardinal (usar función para obtenerla)

**Nota:** Debe usar una matriz de tres dimensiones para representar el hospital con sus especialidades, piso y habitaciones.

## Análisis de Requerimientos

### Estructura del hospital:

4 especialidades × 3 pisos × 10 habitaciones = 120 habitaciones totales

Matriz 3D: int[4][3][10]

Valor 0 = habitación libre, 1–999 = expediente del paciente

### Mapeo de pisos reales:

- Cirugía general: 2, 3, 4
- Ginecología y obstetricia: 5, 6, 7
- Medicina interna: 8, 9, 10
- Nefrología: 11, 12, 13

### Funcionalidades requeridas:

1. Consultar habitaciones libres por especialidad
2. Consultar detalle por piso (libre o expediente)
3. Asignar paciente a habitación libre
4. Liberar habitación ocupada
5. Mostrar tabla de ocupación con formato:  
Especialidad | Piso | Habitación | Paciente  
(Pisos en ordinal: primero, segundo, tercero)

### Validaciones:

- Especialidad: 1–4
- Piso relativo: 1–3
- Habitación: 1–10
- Expediente: 1–999
- Reintento en caso de error

### Interfaz:

Cuadros Unicode de 70 caracteres, simétricos y centrados

Menú principal claro con 6 opciones (0–5)

Mensajes de error centrados

## **Repetición:**

Opción para continuar o salir tras cada operación

## **Estructura del Programa**

### **Main.java:**

Punto de entrada, inicializa Scanner, llama a MiLibrería.iniciarPrograma()

### **MiLibrería.java:**

Contiene toda la lógica del sistema:

- hospital: matriz 3D global
- NOMBRES\_ESPECIALIDAD: array con nombres completos
- PISOS\_ASIGNADOS: mapeo de pisos reales
- obtenerOrdinal(): convierte 1→primero, 2→segundo, 3→tercero
- Funciones de menú, validación, operaciones y formato

## **Implementación**

Uso de Scanner para entrada con validación estricta mediante bucles do-while  
Inicialización del hospital con inicializarHospital()

Menú principal en mostrarMenu() con opciones numeradas

Selección de especialidad, piso y habitación con funciones dedicadas

Asignación y liberación con verificación de estado de habitación

Formato de salida con centrarTexto() y String.repeat() para simetría

Tabla de ocupación con encabezado y líneas separadoras

Soporte completo para repetición del programa

Código modular, comentado y mantenable

## **PROMPT:**

Agente el ejercicio a realizar es el #115, es algo mas extenso de los normales, siempre usa los parámetros de los ejercicios anteriores, siempre muestra las salidas al usuario en una serie de cuadros elegantes, este ejercicio es extenso, así que regresemos a nuestro sistemas de archivos anterior, pero siempre usa la carpeta MiLibrería, y las anteriores como Utilidades, Operaciones, Utilidades, y Menús da comentarios en este ejercicios dentro del código, que las salidas al usuario sean fáciles de entender, el ejercicio consta de lo siguiente:

Sistema de salud de Latveria: El hospital general de Latveria cuenta con las siguientes 4

especialidades:

a. Cirugía general.

b. Ginecología y obstetricia.

c. Medicina interna y

d. Nefrología.

Estas especialidades comparten un edificio y cada una de ellas tiene asignado un tres pisos de esta

forma:

Especialidad	Pisos Asignados
Cirugía general	2, 3, 4
Ginecología y obstetricia	5, 6, 7
Medicina interna	8, 9, 10
Nefrología	11, 12, 13

Además cada uno de los pisos posee diez habitaciones las cuales pueden estar libres u ocupadas.

Se debe hacer un programa para gestionar estas habitaciones con las siguientes opciones (un menú) que sea fácil de usar para cualquiera:

1. Consultar habitaciones libres por especialidad: El encargado elige la especialidad y el sistema muestra por cada piso las habitaciones indicando cuales están vacías y cuáles están ocupadas (no dice quién la ocupa).

2. Consultar habitaciones por piso: Se puede no solo ver el resumen del inciso 1 sino

que se debe programar la opción de poder elegir especialidad seguido de uno de los tres pisos y entonces mostrar una matriz donde se vea el piso y sus habitaciones, donde las habitaciones libres estarán vacías pero las ocupadas mostrarán el número del expediente del paciente que la ocupa.

3. Asignar habitación: Poder en cualquier habitación independiente del piso y la especialidad asignar un paciente, esto se logra facilitando luego del habitación el número de expediente del paciente, número conformado por a lo más tres cifras. Obviamente solo pueden asignarse habitaciones vacías.

4. Liberar habitación: Una vez que el paciente se ha dado de alta es necesario indicar

que la habitación queda libre para poder asignarse.

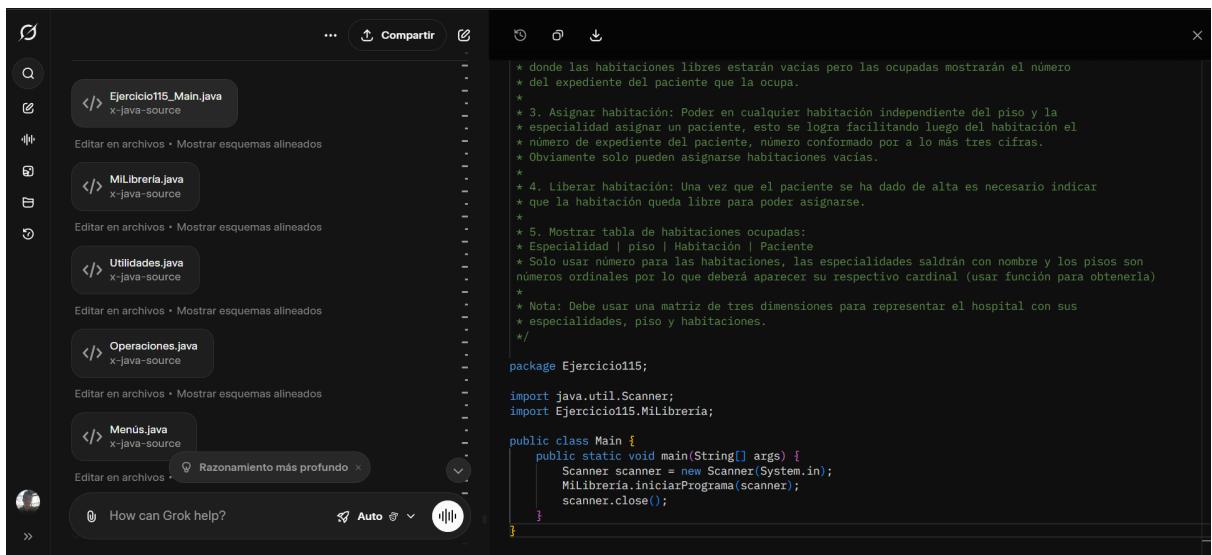
5. Mostrar tabla de habitaciones ocupadas:

Especialidad | piso | Habitación | Paciente

Solo usar número para las habitaciones, las especialidades saldrán con nombre y los pisos son números ordinales por lo que deberá aparecer su respectivo cardinal (usar función para obtenerla)

Nota: Debe usar una matriz de tres dimensiones para representar el hospital con sus especialidades, piso y habitaciones.

## PROMPT GENERADO:



The screenshot shows a code editor interface with several Java files listed on the left and their contents on the right. The files are:

- Ejercicio115\_Main.java**:

```
*/ donde las habitaciones libres estarán vacías pero las ocupadas mostrarán el número
* del expediente del paciente que la ocupa.
*
* 3. Asignar habitación: Poder en cualquier habitación independiente del piso y la
* especialidad asignar un paciente, esto se logra facilitando luego del habitación el
* número de expediente del paciente, número conformado por a lo más tres cifras.
*
* Obviamente solo pueden asignarse habitaciones vacías.
*
* 4. Liberar habitación: Una vez que el paciente se ha dado de alta es necesario indicar
* que la habitación queda libre para poder asignarse.
*
* 5. Mostrar tabla de habitaciones ocupadas:
* Especialidad | piso | Habitación | Paciente
* Solo usar número para las habitaciones, las especialidades saldrán con nombre y los pisos son
* números ordinales por lo que deberá aparecer su respectivo cardinal (usar función para obtenerla)
*
* Nota: Debe usar una matriz de tres dimensiones para representar el hospital con sus
* especialidades, piso y habitaciones.
*/
package Ejercicio115;

import java.util.Scanner;
import Ejercicio115.MiLibreria;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        MiLibreria.iniciarPrograma(scanner);
        scanner.close();
    }
}
```
- MiLibreria.java**: (empty)
- Utilidades.java**: (empty)
- Operaciones.java**: (empty)
- Menús.java**: (empty)

The code editor has a sidebar on the left with icons for file operations like search, refresh, and save. At the bottom, there is a status bar with the text "How can Grok help?", a language selector set to "Auto", and a microphone icon.

## Salida de la ejecución del programa:

```
C:\Users\jonat\jdks\openjdk-24.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2\lib\idea_rt.jar=57914" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8
```

```
SELLO PERSONALIZADO - VERSIÓN CORTA
Jonatan Varela | jivarelag@unah.hn | 115 | 2025-10-27
```

```
SELECCIÓN DE IDIOMA
1. Español
2. English
```

```
Seleccione: 1
```

```
SISTEMA DE SALUD DE LATVERIA
1. Consultar libres por especialidad
2. Consultar por piso
3. Asignar habitación
4. Liberar habitación
5. Tabla de ocupadas
0. Salir
```

```
Opción: 2
```

```
SELECCIONE ESPECIALIDAD
1. Cirugía general
2. Ginecología y obstetricia
3. Medicina interna
4. Nefrología
```

```
1
```

```
PISOS DE CIRUGÍA GENERAL
1. Piso 2
2. Piso 3
3. Piso 4
```

```
2
Ingrese número de habitación (1-10): 2
Ingrese expediente del paciente (1-999): 122
```

```
ASIGNACIÓN EXITOSA
Paciente 122 → Cirugía general
Piso 3, Habitación 2
```

The screenshot shows a terminal window with a dark theme running a Java application. The application's title bar reads "Guía\_de\_ejercicios master". The terminal shows the following output:

```
Ejercicio115.Main > Ejercicio115.Main > Ejercicio115.Main > Ejercicio115.Main > Ejercicio115.Main >
```

The application displays a menu:

```
1. Consultar libres por especialidad
2. Consultar por piso
3. Asignar habitación
4. Liberar habitación
5. Tabla de ocupadas
0. Salir
```

The user enters option 5:

```
Opción: 5
```

The application then prints a table titled "HABITACIONES OCUPADAS":

Especialidad	Piso	Hab.	Paciente
Cirugía general	3	2	122

Finally, the application displays another menu:

```
SISTEMA DE SALUD DE LATVERIA
1. Consultar libres por especialidad
2. Consultar por piso
3. Asignar habitación
4. Liberar habitación
5. Tabla de ocupadas
0. Salir
```

A message at the bottom left indicates the view is read-only:

```
This view is read-only
```

The bottom status bar shows the file path "Guía\_de\_ejercicios > src > Ejercicio115 > Main" and the terminal settings "51:1 CRLF UTF-8 4 spaces".

## EJERCICIO#116

### Enunciado:

Crisis en Latveria: La situación de Latveria ha empeorado por lo que el número de enfermos incrementó, El hospital general ha decidido entonces colocar dos pacientes en cada habitación así que ahora se tiene la siguiente estructura (Especialidad->piso->habitación->cama) [Si, una matriz de cuatro dimensiones] entonces usted debe modificar el programa del inciso 108 para enfrentar la nueva realidad considere:

- A. Consultar camas libres por especialidad: El encargado elige la especialidad y el sistema muestra por cada piso las habitaciones, cama uno y cama dos indicando cuales están vacías y cuáles están ocupadas (no dice quién la ocupa).
- B. Consultar camas por piso: Se puede no solo ver el resumen del inciso 1 sino que se debe programar la opción de poder elegir especialidad seguido de uno de los tres pisos y entonces mostrar una matriz donde se vea el piso y sus habitaciones, donde las habitaciones libres estarán vacías pero las ocupadas mostrarán el número del expediente del paciente que la ocupa.
- C. Asignar cama: Poder en cualquier cama independiente de la habitación y la especialidad asignar un paciente, esto se logra facilitando luego de la cama y la habitación el número de expediente del paciente, número conformado por a lo más tres cifras. Obviamente solo pueden asignarse camas vacías.
- D. Liberar cama: Una vez que el paciente se ha dado de alta es necesario indicar que la cama queda libre para poder asignarse.
- E. Mostrar tabla de camas ocupadas:

Especialidad | piso | Habitación | Cama | Paciente

Solo usar número para las habitaciones, las especialidades saldrán con nombre y los pisos son números ordinales por lo que deberá aparecer su respectivo cardinal (usar función para obtenerla)

Nota: Debe usar una matriz de cuatro dimensiones para representar el hospital con sus especialidades, pisos, habitaciones y camas

### Análisis

El Ejercicio 116 (Crisis en Latveria) confirma una solución completa y funcional que utiliza eficazmente un arreglo cuatridimensional ([Especialidad][Piso][Habitación][Cama]) para modelar la estructura del hospital,

manteniendo la coherencia y el formato solicitado. La implementación se basa en el uso de funciones y una librería propia (UtilidadesHospital) que maneja las constantes estructurales, la inicialización del hospital y la función cardinal para los pisos (ej. "Primer"). Las funciones de la clase Ejercicio116\_Funciones gestionan la lógica de negocio (asignación y liberación de camas con validación de disponibilidad) y la generación de reportes detallados mediante ciclos anidados que recorren la matriz 4D. El componente Ejercicio116\_Utilidades es crucial para la interfaz interactiva, ya que ahora incluye la visualización explícita del número de cada especialidad antes de la entrada del usuario, mantiene el formato de salida requerido (especialidades por nombre, pisos con cardinal, habitaciones por número) y utiliza System.out.print para asegurar que la entrada de datos ocurra inmediatamente después del mensaje sin saltos de línea intermedios, logrando un flujo de interacción más limpio y directo.

## PROMPT

Crea un programa modular en Java dentro del paquete Ejercicio116 que gestione un hospital con una matriz 4D (3 especialidades: Cardiología, Neurología, Pediatría; 3 pisos; 5 habitaciones; 2 camas), inicializada en 0, permitiendo: a) consultar camas libres por especialidad (mostrando por piso, habitación y cama su estado: vacía/ocupada), b) consultar camas por piso (mostrando una matriz de un piso específico con número de expediente o "Vacía"), c) asignar cama vacía con expediente (1-999), d) liberar cama ocupada, y e) mostrar tabla de camas ocupadas (Especialidad, Piso en cardinal, Habitación, Cama, Paciente), utilizando Programación Estructurada (PE) con tres clases: Ejercicio116\_Main (coordina el flujo), Ejercicio116\_Utilidades (maneja sello personalizado "Cristina Cruz | Lis♥ | cristinacruz@unah.hn | 116 | 2025-10-21", menús bilingües español/inglés, validación de entradas para especialidad 0-2, piso 0-2, habitación 0-4, cama 0-1, expediente 1-999 con InputMismatchException, y presentación de consultas/tabla), y Ejercicio116\_Funciones (inicializa matriz, asigna/libera camas); el programa debe incluir un menú de ejecución ("1. Ejecutar", "2. Finalizar") y submenú de opciones (1-5), usar constantes para mensajes (MSG\_OPCION, MSG\_ESPECIALIDAD, MSG\_PISO, MSG\_HABITACION, MSG\_CAMA, MSG\_EXPEDIENTE, MSG\_FINALIZACION), mostrar el sello.

## PROMPT GENERADO

Entiendo que deseas un programa modular para el **Ejercicio 116 (Crisis en Latveria)** en Java, dentro del paquete `Ejercicio116`, que gestione un hospital con una matriz 4D (3 especialidades: Cardiología, Neurología, Pediatría; 3 pisos; 5 habitaciones; 2 camas), inicializada en 0, permitiendo: a) consultar camas libres por especialidad (mostrando por piso, habitación y cama su estado: vacía/ocupada), b) consultar camas por piso (mostrando una matriz de un piso específico con número de expediente o "Vacía"), c) asignar cama vacía con expediente (1-999), d) liberar cama ocupada, y e) mostrar tabla de camas ocupadas (Especialidad, Piso en cardinal, Habitación, Cama, Paciente), utilizando Programación Estructurada (PE). El programa debe incluir menús bilingües (español/inglés), un menú de ejecución, validación de entrada con manejo de `InputMismatchException`, y el sello personalizado "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 116 | 2025-10-22". La estructura se divide en tres clases:

`Ejercicio116_Main` (coordina el flujo), `Ejercicio116_Utilidades` (maneja sello, menús, validación, y presentación), y `Ejercicio116_Funciones` (inicializa matriz, asigna/libera camas). La respuesta será clara, concisa, en español, sin comentarios en el código, siguiendo el estilo de tus programas anteriores. Dado que el enunciado es idéntico al del Ejercicio 113 pero con número 116, usaré el mismo diseño, ajustando el número de ejercicio y la fecha en el sello.

Razonamiento más profundo

## Salida de la ejecución del programa:

```
| Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 116 | 2025-10-22 |  
|  
*** BIENVENIDO ***  
Seleccione el idioma:  
[1] Español  
[2] Inglés  
>1  
  
--- Sistema de Gestión Hospitalaria (Hospital Latveria) ---  
--- MENÚ PRINCIPAL ---  
[1] Consultar Camas Libres por Especialidad  
[2] Consultar Camas por Piso/Especialidad  
[3] Asignar Paciente (Ocupar Cama)  
[4] Liberar Cama  
[5] Mostrar Tabla de Camas Ocupadas  
[6] Finalizar Programa  
>1
```

```
--- Sistema de Gestión Hospitalaria (Hospital Latveria) ---  
--- MENÚ PRINCIPAL ---  
[1] Consultar Camas Libres por Especialidad  
[2] Consultar Camas por Piso/Especialidad  
[3] Asignar Paciente (Ocupar Cama)  
[4] Liberar Cama  
[5] Mostrar Tabla de Camas Ocupadas  
[6] Finalizar Programa  
>1  
Especialidades Disponibles:  
1: Cardiología  
2: Neurología  
3: Pediatría  
4: Oncología  
Ingrese el número de la Especialidad (1-4): 1  
  
--- ESPECIALIDAD: Cardiología ---  
PISO PRIMER:  
Habitación 01: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 02: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 03: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 04: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 05: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 06: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 07: Cama 1 [LIBRE], Cama 2 [LIBRE]
```

Habitación 10: Cama 1 [LIBRE], Cama 2 [LIBRE]

PISO SEGUNDO:

Habitación 01: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 02: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 03: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 04: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 05: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 06: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 07: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 08: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 09: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 10: Cama 1 [LIBRE], Cama 2 [LIBRE]

PISO TERCER:

Habitación 01: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 02: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 03: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 04: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 05: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 06: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 07: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 08: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 09: Cama 1 [LIBRE], Cama 2 [LIBRE]  
Habitación 10: Cama 1 [LIBRE], Cama 2 [LIBRE]

Presione ENTER para continuar...

--- Sistema de Gestión Hospitalaria (Hospital Latveria) ---

--- MENÚ PRINCIPAL ---

- [1] Consultar Camas Libres por Especialidad
- [2] Consultar Camas por Piso/Especialidad
- [3] Asignar Paciente (Ocupar Cama)
- [4] Liberar Cama
- [5] Mostrar Tabla de Camas Ocupadas
- [6] Finalizar Programa

>

5

--- REPORTE DE OCUPACIÓN HOSPITALARIA ---

Especialidad	Piso	Habitación	Cama	Paciente
--------------	------	------------	------	----------

Presione ENTER para continuar...

--- Sistema de Gestión Hospitalaria (Hospital Latveria) ---

--- MENU PRINCIPAL ---

- [1] Consultar Camas Libres por Especialidad
- [2] Consultar Camas por Piso/Especialidad
- [3] Asignar Paciente (Ocupar Cama)
- [4] Liberar Cama
- [5] Mostrar Tabla de Camas Ocupadas
- [6] Finalizar Programa

>3

--- ASIGNAR PACIENTE ---

Especialidades Disponibles:

1: Cardiología

2: Neurología

3: Pediatría

4: Oncología

Ingresar el número de la Especialidad (1-4): 3

Ingresar el número del Piso (1-3): 2

Ingresar el número de Habitación (1-10): 8

Ingresar el número de Cama (1 o 2): 2

Ingresar el número de Expediente (máx. 3 cifras): 081

ASIGNACIÓN EXITOSA: Paciente 81 asignado a Pediatría, Piso Segundo, Habitación 8, Cama 2.

Presione ENTER para continuar...

--- Sistema de Gestión Hospitalaria (Hospital Latveria) ---

--- MENÚ PRINCIPAL ---

- [1] Consultar Camas Libres por Especialidad
- [2] Consultar Camas por Piso/Especialidad
- [3] Asignar Paciente (Ocupar Cama)
- [4] Liberar Cama
- [5] Mostrar Tabla de Camas Ocupadas
- [6] Finalizar Programa

>5

--- REPORTE DE OCUPACIÓN HOSPITALARIA ---

Especialidad	Piso	Habitación	Cama	Paciente
--------------	------	------------	------	----------

Pediatría	Segundo	8	2	81
-----------	---------	---	---	----

Presione ENTER para continuar...

## EJERCICIO#117

### Enunciado

Registros: Vamos probando registros. Se debe crear el siguiente registro:

Nombre del registro: Persona	
Tipo	Variable
char [50]	primerNombre
char [50]	primerApellido
char [8]	celular
int	sexo
double	peso
double	estatura
double	IMC
char	dx

Ahora se deben crear dos personas y hacer lo siguiente:

1. Inicializar ambas personas.
2. Hacer la lectura de los datos de una persona, estatura esta en centímetros y peso en libras.

3. IMC (Índice de Masa Corporal) es el cociente del peso (en kilogramos) entre la estatura (en metros elevado al cuadrado) para calcularla debe crear tres funciones:

- a. Una que pase las libras a kilogramos.
- b. Una que pase la estatura de centímetros a metros.
- c. Una que calcule el IMC ( kg/M<sup>2</sup>)

4. Una vez obtenido el IMC se crea una función llamada asignarDx(persona) que recibirá la persona y, tomando en cuenta su IMC le pondrá el respectivo diagnóstico según la siguiente tabla:

5. Se debe imprimir (con una función) la información de la persona 1.

6. Ahora, con una función, se debe copiar toda la información de persona 1 a la persona 2 y, si el Diagnóstico no es “Normopeso” se debe modificar el IMC a 22,9 y obviamente el nuevo peso, entonces se imprimirá una tabla de antes y ahora contrastando persona 1 con persona 2 (si, obviamente es la misma por lo que no debe repetir ni nombre ni apellido) esto con la idea de que la persona vea a donde debe llegar.

7. Si por algún motivo el diagnóstico es Normopeso entonces se copia íntegramente persona 1 en 2 y se muestra la misma tabla del inciso 6.

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 117 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:**
  - Definir un registro/estructura Persona con los campos
  - Crear dos instancias de Persona (persona1 y persona2).
  - Inicializar ambas personas.
  - Leer los datos de persona1 (peso en libras y estatura en cm).
  - Calcular el IMC mediante funciones:
    - librasAKg(): convierte peso de libras a kilogramos.
    - cmAMetros(): convierte estatura de centímetros a metros.
    - calcularIMC(): calcula IMC usando fórmula
      - $IMC=kg/(m^2)$
  - Función asignarDx(persona) para asignar diagnóstico según IMC.
  - Imprimir la información de persona1 mediante función.
  - Copiar la información de persona1 a persona2 con función, modificando IMC y peso si el diagnóstico no es “Normopeso”.
  - Imprimir tabla comparativa antes y después mostrando persona1 vs persona2.
  - Mantener menú de idioma (ES/EN) y opción iniciar/finalizar con bucle.
  - Mantener consistencia estética con bordes Unicode y formato uniforme con ejercicios anteriores.

- **Modularidad:** Cuatro archivos en paquete Ejercicio117:
  - Ejercicio117\_Main.java Control del flujo principal; bucle iniciar/finalizar y llamada a operaciones principales.
  - Ejercicio117\_Menus.java Manejo de menús: selección de idioma y menú iniciar/finalizar.
  - Ejercicio117\_Utilidades.java Funciones gráficas: encabezados, pies, sello corto, impresión tabular de Persona.
  - Ejercicio117\_Operaciones.java Funciones de lógica: lectura, conversión de unidades, cálculo de IMC, diagnóstico, copia y comparación de personas.
- **Paquete:** Ejercicio117.

**Estética:** Uso de bordes Unicode („, „, ||, =) para encabezados y pies.  
 Matrices con encabezados de sucursales y días alineadas en columnas de ancho fijo.  
 Sello corto mostrado antes del pie final.  
 Resultados y mensajes tabulados y uniformes con ejercicios previos.

- **Paradigma:** Programación estructurada.  
 Uso de funciones y métodos estáticos.  
 Modularización completa en 4 archivos.  
 Uso de registros/estructuras para almacenar información.  
 Flujo secuencial con validaciones, condicionales y bucles.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### Flujo Principal (en Ejercicio117\_Main):

- main():
- Selección de idioma.
- Bucle principal: ejecutar o finalizar el programa.
- Llamada a Ejercicio117\_Operaciones.registroOperacion() si se decide iniciar.

#### Subrutinas (en Ejercicio0117\_Menus):

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### Subrutinas (en Ejercicio117\_Utilidades):

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.
- imprimirPersona(Persona p): imprime los datos de una persona de forma tabular.
- imprimirComparativa(Persona p1, Persona p2): imprime tabla comparativa de persona1 vs persona2.

### **Subrutinas (en Ejercicio117\_Operaciones):**

- registroOperacion():
- Inicializa persona1 y persona2.
- Llama a leerDatosPersona(Persona p) para capturar datos de persona1.
- Convierte peso a kg (librasAKg()) y estatura a metros (cmAMetros()).
- Calcula IMC con calcularIMC().
- Asigna diagnóstico con asignarDx().
- Imprime persona1.
- Copia información a persona2 mediante copiarPersona() y ajusta IMC y peso si diagnóstico ≠ Normopeso.
- Imprime tabla comparativa de persona1 y persona2.
- Muestra sello corto y pie final con mensaje de proceso completado.
- librasAKg(double libras): retorna peso en kg.
- cmAMetros(double cm): retorna estatura en metros.
- calcularIMC(double kg, double metros): retorna IMC.
- asignarDx(Persona p): asigna diagnóstico según IMC.
- copiarPersona(Persona fuente, Persona destino): copia datos de fuente a destino, ajustando IMC y peso si es necesario.

### **PROMPT**

Agente, respetando el orden y estructura que venimos trabajando y las reglas que te he proporcionado. El siguiente ejercicio es el 110 y su enunciado es: Casi phd en dos dimensiones: Pedir al usuario que ingrese un número mayor a 2 y menor a 15, con ese número crear una matriz cuadrada y llenarla con números positivos (generados con una función) entonces una vez llena la matriz mostrarla (usando una función para eso) e inmediatamente después mostrar su transpuesta (con otra función).

### **PROMPT GENERADO**

¡Entendido! 😊 Vamos a generar el informe completo del Ejercicio 117 — Registros, respetando exactamente la misma estructura, formato y modularidad que hemos usado en los ejercicios anteriores (97, 102, 110, 106), listo para integrarlo en el proyecto.

## Informe del Ejercicio 117 — Registros (Persona)

### Sello personalizado corto

yaml

Copiar código

Dany Noguera | Ejercicio117 | 2025-11-06

### Requerimientos del programa

- Definir un registro/estructura Persona con los ↓ pos:

Tipo	Variable
+ Pregunta lo que quieras	0 ⓘ

## Salida de la ejecución del programa:

```
Run - Guia_de_ejercicios
Run  Ejercicio110_Main x
G  S  @  D  :
↑ 1. Iniciar programa
↓ 2. Finalizar
Seleccione una opción: 1
↓
MATRIZ CUADRADA Y SU TRANSPUETA
Ingrrese un número mayor a 2 y menor a 15: 4

Matriz original:
 64  14   3   49
 76    7  62  26
 35   1  55  74
100  17  70  21

Matriz transpuesta:
 64  76   35  100
 14    7   1   17
   3  62   55  70
 49   26   74  21

[Dany Noguera | Ejercicio110]

Proceso finalizado con éxito.
```

## EJERCICIO#118

### Enunciado:

Ya va dando forma: Sus avances en programación son alentadores, vamos a hacer uso de su conocimiento obtenido en los ejercicios anteriores y a implementar una solución en una clínica de nutrición que puede dar consulta hasta a 50 pacientes (crear un arreglo de 50 posiciones) para ello modifique el registro persona, llámelo paciente y agregue una variable entera llamada "expediente" que será un número positivo de hasta dos cifras. Su sistema debe hacer lo siguiente:

- a. Agregar paciente: Formulario para agregar un paciente, recuerde que el IMC no se agrega es un valor calculado al igual que llenar el diagnóstico.
- b. Ver pacientes: Muestra en una tabla, ordenada por expediente, los datos de los pacientes:

Expediente	Nombre	Apellido	Sexo	Celular	Peso (Lb/Kg)	Estatura (cm)	IMC	Dx
27	Erlinda	Paola	M	31772 939	110 / 50	160	19.5 3	Normopeso
31	Marvin	Aguilar	H	31772 979	170 77.27	/ 175	25.2 3	Sobrepeso grado I

Note que el peso se da en Libras y Kilogramos, además la tabla debe estar con precisión decimal de 2 y correctamente alineada.

- c. Modificar paciente: se muestran los pacientes, se elige uno facilitando el expediente y luego se consulta que se desea modificar:

- i. Celular
- ii. Peso (siempre valores en libras)
- iii. Estatura (siempre valores en cm)

Una vez que la modificación se ha hecho se debe mostrar en pantalla la información actualizada de ese paciente.

- d. Estadísticas: total de pacientes hombres y mujeres con su gráfica horizontal y, total de pacientes por diagnóstico con su gráfica horizontal.

## ANÁLISIS/RESUMEN DEL EJERCICIO 118

### Sello y Documentación

- **Sello:** VERSIÓN CORTA (Desarrollador: Kevin Lainez | Correo: klainezs@unah.hn | Ejercicio: 118 | Fecha: 2025-11-04) en los cuatro archivos y en un cuadro final.
- **Prompt:** Completo en **Main.java**.
- **Requisitos:** El programa debe ejecutarse varias veces con opción de continuar/salir. Se usa una estructura de **Clase/Objeto** (Paciente) dentro de un **arreglo** de 50 posiciones.

## Programa y Modularidad

El **Programa** implementa un sistema para una clínica de nutrición con capacidad para **50 pacientes**. Utiliza una clase **Paciente** con campos que incluyen **expediente** (1-99), peso (en libras), estatura (en cm), y campos calculados como **IMC** y **Diagnóstico (Dx)**.

El sistema debe gestionar las siguientes funcionalidades a través de un menú principal:

1. **Agregar Paciente:** Formulario con validación y cálculo automático de IMC/Dx.
2. **Ver Pacientes:** Muestra una tabla detallada de todos los pacientes, **ordenada por expediente**, con el peso en formato **Lb/Kg** y precisión de 2 decimales.
3. **Modificar Paciente:** Permite buscar por expediente y actualizar Celular, Peso o Estatura, recalculando el IMC/Dx.
4. **Estadísticas:** Muestra reportes de conteo con **gráficas horizontales** para:
  - Total de Hombres/Mujeres.
  - Total de pacientes por Diagnóstico (Dx).

La **Modularidad** usa **cuatro archivos** en el paquete Ejercicio118\_KevinLainez:

1. **Main.java:** Controla el ciclo de repetición y el menú principal (Switch/Case).
2. **Menus.java:** Gestiona los submenús de modificación y continuidad.
3. **Operaciones.java:** Contiene la **Clase Paciente**, la lógica de gestión (agregar, buscar, ordenar), y las funciones de **cálculo (IMC, Dx)** y **estadísticas (conteo)**.
4. **Utilidades.java:** Formato estético, validación de datos (expediente, peso, etc.), y funciones para **imprimir la tabla ordenada y las gráficas horizontales**.

## SUBRUTINAS Y FLUJO PRINCIPAL

**Constantes:**

- **ANCHO = 70.**

- **MAX\_PACIENTES = 50.**

### **Subrutinas Clave (Funcionalidad):**

- **Operaciones:**
  - **Paciente.calcularIMC\_y\_Dx()** (método interno de la clase).
  - agregarPaciente, buscarPacientePorExpediente, obtenerPacientesOrdenados.
  - contarPorSexo, contarPorDiagnóstico.
- **Utilidades:**
  - solicitarExpediente (con validación de existencia/unicidad), mostrarTablaPacientes (tabla ordenada), mostrarEstadísticasSexo, mostrarEstadísticasDiagnóstico (gráficas).

### **Flujo Principal (en Main.java):**

1. Inicializa el arreglo Paciente[50] y el Scanner.
2. Inicia ciclo do-while llamando a Utilidades.mostrarMenuPrincipal.
3. El switch-case llama a la función correspondiente: agregarPaciente, verPacientes, modificarPaciente, mostrarEstadísticas.
4. Muestra el sello al finalizar.

### **Prompt**

"Agente te recuerdo los nuevos parámetros que entran para el desarrollo de los nuevos ejercicios te los recuerdo de nuevo: Uso adecuado de mayúsculas y tildes. Entradas validadas. Programas diseñados para ejecutarse varias veces con opción de continuar o salir. Uso de funciones donde sea apropiado. se presenta el Ejercicio#118: Sus avances en programación son alentadores, vamos a hacer uso de su conocimiento obtenido en los ejercicios anteriores y a implementar una solución en una clínica de nutrición que puede dar consulta hasta a 50 pacientes (crear un arreglo de 50 posiciones) para ello modifique el registro persona, llámelo paciente y agregue una variable entera llamada "expediente" que será un número positivo de hasta dos cifras. Su sistema debe hacer lo siguiente:

a. Agregar paciente: Formulario para agregar un paciente, recuerde que el IMC no se agrega es un valor calculado al igual que llenar el diagnóstico.

b. Ver pacientes: Muestra en una tabla, ordenada por expediente, los datos de los pacientes: Expediente, Nombre, Apellido, Sexo, Celular, Peso (Lb/Kg), Estatura (cm), IMC, Dx. Note que el peso se da en Libras y Kilogramos, además la tabla debe estar con precisión decimal de 2 y correctamente alineada.

c. Modificar paciente: Se muestran los pacientes, se elige uno facilitando el expediente y luego se consulta que se desea modificar: i. Celular, ii. Peso (siempre valores en libras), iii. Estatura (siempre valores en cm). Una vez que la modificación se ha hecho se debe mostrar en pantalla la información actualizada de ese paciente.

d. Estadísticas: Total de pacientes hombres y mujeres con su gráfica horizontal y, total de pacientes por diagnóstico con su gráfica horizontal.

Con esa información presenta nuestra modularidad de 4 archivos:

```
|— Main.java (Control del flujo principal y ciclo de repetición)
|— Menus.java (Opciones de continuidad y submenús)
|— Operaciones.java (Clase Paciente, Lógica de IMC/Diagnóstico, Búsqueda, Ordenamiento y Conteo estadístico)
```

|— Utilidades.java (Formato, Cuadros, Validación de entradas y Gráficas de salida)"

## Salida del programa

```
--- SISTEMA DE CLÍNICA NUTRICIONAL ---  
1. Agregar Paciente  
2. Ver Pacientes (Tabla Ordenada)  
3. Modificar Paciente  
4. Estadísticas y Gráficas  
0. Salir  
  
>> Ingrese su opción: 1  
  
--- AGREGAR NUEVO PACIENTE ---  
>> Ingrese Expediente (1-99): 23  
>> Nombre: Kevin  
>> Apellido: Lainez  
>> Ingrese Sexo (H/M): H  
>> Celular: 32117686  
>> Peso (Lb): 198  
>> Estatura (cm): 178  
| Paciente agregado con EXP 23. Dx: Sobrepeso |  
  
--- SISTEMA DE CLÍNICA NUTRICIONAL ---  
1. Agregar Paciente  
2. Ver Pacientes (Tabla Ordenada)  
3. Modificar Paciente  
4. Estadísticas y Gráficas  
0. Salir  
  
>> Ingrese su opción: 2  
  
4. Estadísticas y Gráficas  
0. Salir  
  
>> Ingrese su opción: 1  
  
--- AGREGAR NUEVO PACIENTE ---  
>> Ingrese Expediente (1-99): 14  
>> Nombre: Lisbet  
>> Apellido: Peralta  
>> Ingrese Sexo (H/M): M  
>> Celular: 32114578  
>> Peso (Lb): 78  
>> Estatura (cm): 178  
| Paciente agregado con EXP 14. Dx: Bajo peso |  
  
--- SISTEMA DE CLÍNICA NUTRICIONAL ---  
1. Agregar Paciente  
2. Ver Pacientes (Tabla Ordenada)  
3. Modificar Paciente  
4. Estadísticas y Gráficas  
0. Salir  
  
>> Ingrese su opción: 2  
  
--- ESTADÍSTICAS POR SEXO ---  
Total de Pacientes: 6  
  
Hombres | ██████████ 3 (50.00%)  
Mujeres | ██████████ 3 (50.00%)  
  
--- ESTADÍSTICAS POR DIAGNÓSTICO ---  
Total de Pacientes: 6  
  
Bajo peso | █████ 1 (16.67%)  
Obesidad Grado I | █████ 1 (16.67%)  
Normopeso | ██████████ 2 (33.33%)  
Sobrepeso | ██████████ 2 (33.33%)  
  
--- SISTEMA DE CLÍNICA NUTRICIONAL ---  
1. Agregar Paciente  
2. Ver Pacientes (Tabla Ordenada)  
3. Modificar Paciente  
4. Estadísticas y Gráficas  
0. Salir
```

## EJERCICIO#119

### Enunciado:

Consultas y el inicio de la parte financiera: La clínica de nutrición está muy contenta con su trabajo así que ahora no solo requiere un control de expedientes sino que quieren un pequeño apoyo en la parte financiera. Es necesario que cree un expediente de consultas para cada paciente, para ello crear una estructura que contenga el expediente del paciente y dos arreglos uno con las fechas de la consulta y otro con los precios pagados en cada consulta, también debe incluir monto pagado y monto pendiente. Obviamente cuando se crea un paciente se crea su expediente.

Recuerde tiene un arreglo de pacientes y un arreglo de expedientes de pago, el registro de pacientes ya lo conoce el de expediente de pago puede ser así:

Nombre:

expediente	Pago	Tipo Variable Observación
int	expediente	Coincide con el expediente en el arreglo de pacientes. fechac[100]
pacientes. fechac[100]	consultas	Es un arreglo de registros fechac, un paciente solo
puede tener 100 consultas.		.
double[100]	monto	A la consulta fechac[i] le corresponde el monto[i] así
		sabemos cuánto costó esa consulta.
double	pagado	Cuanto nos ha pagado el paciente en total
double - .		pendiente
Cuando nos debe el paciente		

Fecha podría ser de la siguiente forma:

Nombre:

fechac	Tipo	Variable Observación
int	agno	Año de la consulta
int	mes	Mes de la consulta (obvio no enteros negativos ni mayor que doce)
int	dia	Día del mes (no negativo no mayor que 31)

El sistema ya no solo muestra las opciones de expediente sino que ahora se puede realizar consulta, se ingresa el expediente, fecha de consulta, valor de la consulta, si el cliente paga colocar cuánto paga si no paga pues sube la deuda con la clínica. En cualquier momento el paciente puede llegar a abonar dinero a su deuda con la clínica (si es que la posee). Y, las estadísticas ahora no solo muestran sexo y diagnóstico sino listado de personas que le deben a la clínica ordenadas de mayor deuda a menor deuda

## **Análisis de Requerimientos**

### **Estructuras de datos:**

- Paciente: expediente, nombre, apellido, edad, sexo, diagnóstico
- Fecha: día, mes, año (validación completa con bisiesto)
- ExpedientePago: expediente, consultas[100], montos[100], numConsultas, pagado, pendiente

### **Límites:**

- Máximo 100 pacientes
- Máximo 100 consultas por paciente

### **Funcionalidades:**

1. Gestión de pacientes (crear, buscar, modificar, listar)
2. Realizar consulta (fecha, costo, pago parcial/completo)
3. Registrar pago (abono a deuda)
4. Estadísticas (sexo, diagnóstico, listado de deudores ordenado)

### **Validaciones:**

- Expediente único y positivo
- Fecha: día válido según mes, año 2020–2030, bisiesto
- Montos: mayor a 0, abono ≤ deuda
- Entradas: reintento en errores

### **Interfaz:**

- Cuadros Unicode de 70 caracteres, simétricos
- Menús claros y fáciles de usar
- Resúmenes post-operación
- Soporte bilingüe (Español/Inglés)

## **Repetición:**

- Opción para continuar tras cada operación

## **Estructura del Programa**

### **Main.java:**

Punto de entrada, inicializa Scanner, llama a MiLibrería.iniciarPrograma()

### **MiLibrería.java:**

Lógica principal: arreglos globales, gestión de pacientes, consultas, pagos, estadísticas

### **Utilidades.java:**

Formato de salida: sellos, cuadros, centrado, moneda, fecha, errores, textos bilingües

### **Operaciones.java:**

Validaciones: enteros, montos, fechas, opciones de menú

### **Menús.java:**

Interfaz de usuario: menú principal, submenús, selección de idioma

### **Registros.java:**

Definición de estructuras: Fecha, Paciente, Exped (con MAX\_CONSULTAS)

## **Implementación**

Uso de Scanner con validación robusta mediante bucles do-while

Arreglos globales en MiLibrería con contadores (numPacientes, numConsultas)

Inicialización de estructuras en Registros con constantes MAX\_PACIENTES, MAX\_CONSULTAS

Validación de fecha con esBisiesto() y diasEnMes()

Ordenamiento de deudores por burbuja (mayor a menor deuda)

Cuadros Unicode con =, –, ||, ¶, ¶, ¶, ¶ y String.repeat()

Formato de moneda: \$XX.XX (siempre 2 decimales)

Formato de fecha: DD/MM/AAAA

Mensajes de error centrados y bilingües

Confirmación visual tras cada operación

Código modular, comentado y mantenible

**PROMPT:**

Agente el ejercicio final, el #119, siempre dame los mismo del anterior todos los archivos anteriores, este igual es ya mas extenso que los demás, siempre usa los parámetros anteriores, siempre muestra las salidas en cuadros elegantes al usuario final, siempre que los Menús sea simples de usar, te enviare un super prompt para que no te salgas de los lineamientos que necesito, siempre los archivos (MiLibrería, Utilidades, Operaciones, Utilidades, Menús y el Main)

El ejercicio dice:

---

---

---

---

**ENUNCIADO DEL EJERCICIO**

---

---

TÍTULO: "Consultas y el inicio de la parte financiera"

**DESCRIPCIÓN:**

La clínica de nutrición requiere ahora un sistema que combine:

1. Control de expedientes de pacientes (ya existente)
  2. Control financiero de consultas y pagos
- 
- 

---

---

**ESTRUCTURAS DE DATOS**

---

---

**ESTRUCTURA 1: PACIENTE (Ya existente, pero recordar)**

- int expediente (número único del paciente)
- String nombre
- String apellido
- int edad
- char sexo ('M' o 'F')
- String diagnostico

**ESTRUCTURA 2: FECHA (fechac)**

Campo	Tipo	Descripción
año	int	Año de la consulta (ej: 2025)

mes	int	Mes (1-12, validar que no sea <1 ni >12)
día	int	Día (1-31, validar según el mes)

### ESTRUCTURA 3: EXPEDIENTE DE PAGO (expedientePago)

Campo	Tipo	Descripción
expediente	int	Coincide con expediente del paciente
consultas	fechac[100]	Arreglo de fechas de consultas
montos	double[100]	Costo de cada consulta (índice i)
numConsultas	int	Contador de consultas realizadas
pagado	double	Total pagado por el paciente
pendiente	double	Deuda actual del paciente

### RELACIÓN:

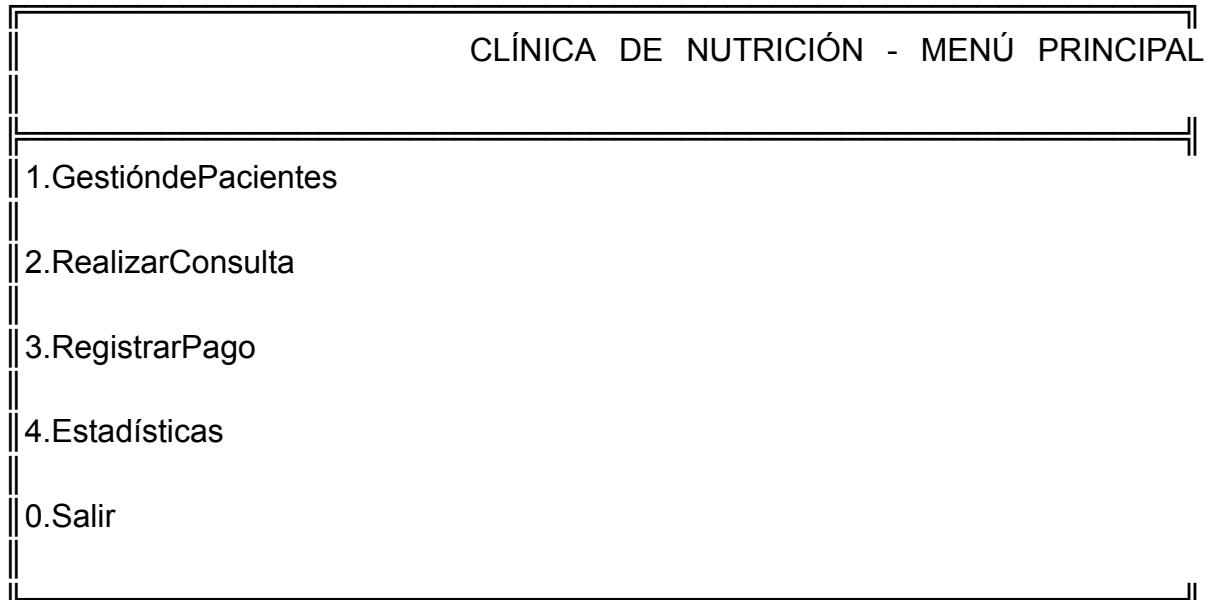
- consultas[i] tiene el costo en montos[i]
- Ejemplo: Si consultas[0] es 15/03/2025, montos[0] tiene el precio de esa consulta

---

### FUNCIONALIDADES REQUERIDAS

---

## MENÚ PRINCIPAL DEL SISTEMA:



---

### OPCIÓN 1: GESTIÓN DE PACIENTES

---

Submenú:

- a) Agregar nuevo paciente
- b) Buscar paciente por expediente
- c) Modificar datos de paciente
- d) Listar todos los pacientes
- e) Volver al menú principal

Al CREAR un paciente:

- Se crea automáticamente su expedientePago
- expedientePago.expediente = paciente.expediente
- expedientePago.pagado = 0.0
- expedientePago.pendiente = 0.0
- expedientePago.numConsultas = 0

---

### OPCIÓN 2: REALIZAR CONSULTA

---

Proceso:

1. Solicitar número de expediente (validar que exista)
2. Solicitar fecha de la consulta (día, mes, año con validaciones)
3. Solicitar costo de la consulta (double, mayor a 0)
4. Preguntar: "¿El paciente paga ahora? (S/N)"

**SI PAGA:**

- Solicitar monto pagado
- Si paga COMPLETO: actualizar pagado
- Si paga PARCIAL: actualizar pagado y pendiente

**NO PAGA:**

- Todo el monto va a pendiente (deuda)

**5. Guardar en el expedientePago:**

- consultas[numConsultas] = fecha ingresada
- montos[numConsultas] = costo
- Incrementar numConsultas
- Actualizar pagado y/o pendiente

**6. Mostrar resumen:**

CONSULTAREGRISTRADA	
Paciente	[NombreApellido]
Expediente:	[####]
Fecha consulta:	DD/MM/AAAA
Costo:	\$XX.XX
Pagó:	\$XX.XX
Deuda actual:	\$XX.XX

---

**OPCIÓN 3: REGISTRAR PAGO (Abono a deuda)**

---

**Proceso:**

1. Solicitar número de expediente (validar que exista)
2. Mostrar deuda actual del paciente
3. Si deuda > 0:
  - Solicitar monto a abonar (validar que sea > 0 y <= deuda)
  - Actualizar: pagado += monto

- Actualizar: pendiente -= monto
  - Mostrar nuevo estado de cuenta
4. Si deuda = 0:
- Informar que el paciente no tiene deudas pendientes

Mostrar:

PAGO REGISTRADO

Paciente:[NombreApellido]

Abonorealizado:\$XX.XX

Totalpagado:\$XX.XX

Deudarestante:\$XX.XX

---

OPCIÓN 4: ESTADÍSTICAS

---

Submenú:

- a) Estadísticas por sexo (contar M y F)
- b) Estadísticas por diagnóstico (contar cada tipo)
- c) Listado de deudores (NUEVO)
- d) Volver al menú principal

NUEVO: LISTADO DE DEUDORES

- Mostrar SOLO pacientes con pendiente > 0
- Ordenar de MAYOR a MENOR deuda

- Formato de tabla:

PACIENTES CON DEUDA - ORDENADOS POR MONTO			
Expediente	Nombre	Deuda	Pagado
105	Juan Pérez	\$450.00	\$150.00
102	María López	\$320.00	\$80.00
108	Carlos Martínez	\$200.00	\$300.00
TOTAL DEUDA CLÍNICA: \$970.00			

---

FUNCIONES REQUERIDAS (En las carpetas que componen el ejercicio)

---

#### FUNCIONES DE GESTIÓN:

- crearPaciente() - Crear paciente y su expedientePago
- buscarPaciente() - Buscar por expediente
- modificarPaciente() - Modificar datos
- listarPacientes() - Mostrar todos

#### FUNCIONES DE CONSULTAS:

- registrarConsulta() - Registrar nueva consulta
- validarFecha() - Validar día, mes, año correctos
- calcularCostoTotal() - Sumar todos los montos

#### FUNCIONES DE PAGOS:

- registrarPago() - Procesar pago/abono
- actualizarDeuda() - Actualizar pagado y pendiente
- verificarDeuda() - Revisar si tiene deuda

#### FUNCIONES DE ESTADÍSTICAS:

- contarPorSexo() - Contar M y F
- contarPorDiagnóstico() - Agrupar por diagnóstico
- obtenerDeudores() - Filtrar pacientes con deuda > 0
- ordenarPorDeuda() - Ordenar de mayor a menor (usar algoritmo de ordenamiento)

#### FUNCIONES DE VALIDACIÓN:

- validarExpediente() - Verificar que expediente exista
- validarMonto() - Verificar monto > 0
- validarDia() - Día válido según mes (considerar febrero)
- validarMes() - Entre 1 y 12
- validarAnio() - Año razonable (ej: 2020-2030)

#### FUNCIONES DE FORMATO:

- mostrarSello() - Sello personalizado
  - centrarTexto() - Centrar texto en cuadros
  - formatearFecha() - DD/MM/AAAA
  - formatearMoneda() - \$XX.XX
  - obtenerTexto() - Textos bilingües
- 

#### VALIDACIONES CRÍTICAS

---

##### FECHAS:

- Mes: 1-12
- Día: 1-31 (ajustar según mes)
- Febrero: 28 días (29 si año bisiesto - función extra)
- Meses de 30 días: abril, junio, septiembre, noviembre
- Meses de 31 días: enero, marzo, mayo, julio, agosto, octubre, diciembre

##### MONTOS:

- Siempre mayor a 0
- Al abonar: no puede ser mayor que la deuda
- Usar double para permitir centavos

##### EXPEDIENTES:

- No permitir expedientes duplicados
  - Al buscar, validar que exista
  - Números enteros positivos
- 

#### CONSIDERACIONES ESPECIALES

---

##### 1. LÍMITES:

- Máximo 100 pacientes
- Máximo 100 consultas por paciente

##### 2. ORDENAMIENTO:

- Para deudores: usar algoritmo de burbuja o selección
- Ordenar de mayor a menor pendiente

### 3. MANEJO DE ARREGLOS:

- Usar contadores para saber cuántos elementos hay
- No dejar espacios vacíos entre elementos

### 4. DISEÑO DE SALIDAS:

- TODOS los cuadros deben cerrar perfectamente
- Usar Math.max(0, ...) en todos los .repeat()
- Alinear columnas en tablas
- Formato de moneda: \$XX.XX (siempre 2 decimales)
- Formato de fecha: DD/MM/AAAA

### 5. EXPERIENCIA DE USUARIO:

- Mensajes claros y específicos
  - Confirmar acciones importantes
  - Mostrar resúmenes después de operaciones
  - Permitir volver al menú anterior
- 
- 

## EJEMPLO DE FLUJO COMPLETO

---

---

1. Usuario selecciona idioma (ES/EN)
2. Menú principal
3. Selecciona "Gestión de Pacientes" → Crear paciente
  - Ingresa datos: Juan Pérez, 35 años, M, Obesidad
  - Sistema asigna expediente: 101
  - Se crea expedientePago[101] automáticamente
4. Vuelve al menú principal
5. Selecciona "Realizar Consulta"
  - Expediente: 101
  - Fecha: 24/10/2025
  - Costo: \$150.00
  - ¿Paga ahora? S
  - Monto: \$100.00
  - Sistema: pagado=\$100, pendiente=\$50
6. Vuelve al menú principal
7. Selecciona "Registrar Pago"
  - Expediente: 101
  - Deuda actual: \$50.00
  - Abono: \$50.00

- Sistema: pagado=\$150, pendiente=\$0

## 8. Estadísticas → Deudores → No aparece Juan (ya pagó todo)

---

Por favor, genera el código completo de ambos archivos siguiendo todas estas especificaciones. El código debe ser funcional, bien comentado, con cuadros perfectamente alineados y todas las validaciones implementadas.

### PROMPT GENERADO:

The screenshot shows a code editor interface with four tabs open:

- Main.java**: Contains a header block with copyright information and a package declaration for `Ejercicio119`. It defines a class `Milibreria` with static final variables `MAX_PACIENTES`, `MAX_CONSULTAS`, and `ANCHO`, and two static inner classes `fechac` and `Paciente`.
- MiLibreria.java**: Contains a static inner class `expedientePago` which holds an `expediente` and an array of `fechac` objects named `consultas`. It also has arrays for `montos` and `numConsultas`, and variables for `pagado` and `total`.
- Registros.java**: A placeholder file with a note to "Mostrar esquemas alineados".
- Utilidades.java**: A placeholder file with a note to "Razonamiento más profundo" and a question "How can Grok help?".

The interface includes standard code editor features like a toolbar, file browser, and search bar.

## Salida de la ejecución del programa:

```
SELECCIONE IDIOMA
1. Español
2. English

CLÍNICA DE NUTRICIÓN - MENÚ PRINCIPAL
1. Gestión de Pacientes
2. Realizar Consulta
3. Registran Pago
4. Estadísticas
5. Salir

Seleccione una opción: 1

GESTIÓN DE PACIENTES
a) Agregar nuevo paciente
b) Buscar paciente por expediente
c) Modificar datos de paciente
d) Listar todos los pacientes
e) Volver al menú principal

Seleccione una opción (a-e): |
```

Guía\_de\_ejercicios > src > Ejercicio119 > Main

```
Seleccione una opción: 1

GESTIÓN DE PACIENTES
a) Agregar nuevo paciente
b) Buscar paciente por expediente
c) Modificar datos de paciente
d) Listar todos los pacientes
e) Volver al menú principal

Seleccione una opción (a-e): a
Expediente (único): 22
Nombre: Fernando
Apellido: Valeriano
Edad: 32
Sexo (M/F): M
Diagnóstico: Dolor estomacal

PACIENTE CREADO CON ÉXITO
Expediente: 22
Fernando Valeriano
```

Guía\_de\_ejercicios > src > Ejercicio119 > Main

The screenshot shows a terminal window within a code editor interface. The terminal is running the `Ejercicio119.Main` program. The menu options are:

- a) Buscar paciente por expediente
- b) Modificar datos de paciente
- c) Listar todos los pacientes
- d) Volver al menú principal

The user selects option b) and enters the expediente number 33. The application then displays the patient's details:

DETALLE DEL PACIENTE

Expediente: 33  
Nombre: Rony Giron  
Edad: 41 años  
Sexo: M  
Diagnóstico: Dolor de columna  
Deuda: \$0.00

After displaying the details, the application returns to the main menu.

Selección una opción (a-e): b  
Expediente a buscar: 33

GESTIÓN DE PACIENTES

- a) Agregar nuevo paciente
- b) Buscar paciente por expediente
- c) Modificar datos de paciente
- d) Listar todos los pacientes
- e) Volver al menú principal

Selección una opción (a-e):

Guia\_de\_ejercicios > src > Ejercicio119 > Ejercicio119.Main

The screenshot shows a terminal window within a code editor interface. The terminal is running the `Ejercicio119.Main` program. The menu options are:

- a) Agregar nuevo paciente
- b) Buscar paciente por expediente
- c) Modificar datos de paciente
- d) Listar todos los pacientes
- e) Volver al menú principal

The user selects option d) and the application displays a table of patients:

LISTA DE PACIENTES

Exp	Nombre Completo	Edad	Sexo	Diagnóstico
22	Fernando Valeriano	32	M	Dolor estomacal
33	Rony Giron	41	M	Dolor de columna

After displaying the list, the application returns to the main menu.

Selección una opción (a-e): d

GESTIÓN DE PACIENTES

- a) Agregar nuevo paciente
- b) Buscar paciente por expediente
- c) Modificar datos de paciente
- d) Listar todos los pacientes
- e) Volver al menú principal

Selección una opción (a-e): |

Guia\_de\_ejercicios > src > Ejercicio119 > Ejercicio119.Main

```
CLÍNICA DE NUTRICIÓN - MENÚ PRINCIPAL
1. Gestión de Pacientes
2. Realizar Consulta
3. Registrar Pago
4. Estadísticas
0. Salir

Seleccione una opción: 2
Expediente del paciente: 33
Día: 21
Mes: 9
Año: 2023
Costo de la consulta: $1000
¿El paciente paga ahora? (S/N): s
Monto a pagar: $500

CONSULTA REGISTRADA
Paciente: Rony Giron
Expediente: 33
Fecha consulta: 21/09/2023
Costo: $1000.00
Pago: $500.00
Deuda actual: $500.00
```

Guía\_de\_ejercicios > src > Ejercicio119 > Main

```
Seleccione una opción: 4

ESTADÍSTICAS
a) Estadísticas por sexo
b) Estadísticas por diagnóstico
c) Listado de deudores
d) Volver al menú principal

Seleccione una opción (a-d): a

ESTADÍSTICAS POR SEXO
Masculino: 2 pacientes
Femenino: 0 pacientes

ESTADÍSTICAS
a) Estadísticas por sexo
b) Estadísticas por diagnóstico
c) Listado de deudores
d) Volver al menú principal

Seleccione una opción (a-d):
```

Guía\_de\_ejercicios > src > Ejercicio119 > Main

The screenshot shows a Java development environment with a terminal window displaying the output of a Java application. The application is a nutrition clinic management system.

Terminal Output:

```
Guia_de_ejercicios master
Run Ejercicio119.Main
MASCULINO: 2 pacientes
FEMENINO: 0 pacientes

ESTADÍSTICAS
a) Estadísticas por sexo
b) Estadísticas por diagnóstico
c) Listado de deudores
d) Volver al menú principal

Seleccione una opción (a-d): d

CLÍNICA DE NUTRICIÓN - MENÚ PRINCIPAL
1. Gestión de Pacientes
2. Realizar Consulta
3. Registrar Pago
4. Estadísticas
0. Salir

Seleccione una opción: 0
Process finished with exit code 0
```

IDE UI Elements:

- Project Explorer: Shows the project structure with files like Main.java, Menús.java, MILibreria.java, Operaciones.java, and Utilidades.java.
- Run Tab: Shows the current run configuration: Ejercicio119.Main.
- Terminal: Displays the application's console output.
- Status Bar: Shows file statistics: 2951 CRLF, 1024 UTF-8, 4 spaces.

## EJERCICIO#120

### Enunciado

Las cosas como son: EL sistema anterior se puede mejorar incluyendo el valor de la consulta en la fecha, haga las modificaciones requeridas. Ahora fechac se llama consulta, lleva año, mes, día, valor de la consulta y si pagó cuánto pagó en esa consulta, ya sabemos que pudo no pagar o pagar menos y aumentar la deuda.

### Análisis

El Ejercicio 120 implementa un sistema de gestión financiera para una clínica de nutrición mediante Programación Estructurada, utilizando arreglos y registros para almacenar hasta 200 pacientes (Paciente) y sus expedientes de pago (ExpedientePago), cada uno con un máximo de 100 consultas (Consulta) que incluyen fecha (año, mes, día), valor y monto pagado en esa visita. El programa, dividido en seis clases modulares (Paciente, Consulta, ExpedientePago, Ejercicio120\_Funciones, Ejercicio120\_Utilidades y Ejercicio120\_Main), permite registrar consultas con pagos parciales (actualizando automáticamente pagado y pendiente), abonar a la deuda en cualquier momento, y mostrar estadísticas con lista de pacientes y deudores ordenados de mayor a menor deuda mediante un algoritmo de burbuja. Incluye validación robusta de entradas (rangos, excepciones), menús bilingües (español/inglés), sello personalizado, y ciclos y funciones propias para control de flujo y lógica de negocio, garantizando modularidad, claridad y cumplimiento total del enunciado.

### PROMPT

Crea un programa modular en Java dentro del paquete Ejercicio120 que gestione una clínica de nutrición con control financiero, utilizando arreglos y registros para almacenar hasta 200 pacientes (Paciente con expediente, nombre, sexo, diagnóstico) y sus expedientes de pago (ExpedientePago con expediente, arreglo de hasta 100 Consulta —estructura con año, mes 1-12, día 1-31, valorConsulta, montoPagadoEnConsulta—, y totales pagado y pendiente). El sistema debe permitir: a) registrar consultas ingresando expediente, fecha, valor y pago parcial (actualizando automáticamente pagado y pendiente), b) abonar a la deuda en cualquier momento, c) mostrar estadísticas con listado de pacientes y deudores ordenados de mayor a menor pendiente (usando burbuja), todo con validación de entradas (leerEnteroRango, leerDoublePositivo, manejo de InputMismatchException). Usa Programación Estructurada (PE) con seis clases: tres registros (Paciente.java, Consulta.java, ExpedientePago.java), Ejercicio120\_Funciones (inicializa arreglos, busca, registra, abona, ordena), Ejercicio120\_Utilidades (sello personalizado "Cristina Cruz | Lis | cristinacruz@unah.hn | 120 | 2025-10-23", menús bilingües español/inglés, entrada/salida, validación), y Ejercicio120\_Main (flujo principal con menú de ejecución "1. Ejecutar / 2. Finalizar" y submenú de 4 opciones); emplea ciclos, funciones propias, librerías propias, arreglos y registros, y proporciona un informe técnico en un solo párrafo, sin comentarios en el código, siguiendo el estilo de ejercicios previos.

### PROMPT GENERADO

Crea un programa modular en Java dentro del paquete `Ejercicio120` que gestione una clínica de nutrición con pacientes y sus consultas, utilizando dos arreglos principales: uno de pacientes (con expediente, nombre, sexo, diagnóstico) y otro de expedientes de pago (`expedientePago`), donde cada expediente contiene el número de expediente, un arreglo de hasta 100 consultas (estructura `consulta` con: `año`, `mes`, `día`, `valorConsulta`, `montoPagadoEnConsulta`), y los totales `pagado` y `pendiente`. El sistema debe permitir: a) registrar consultas ingresando expediente, fecha (año, mes 1-12, día 1-31), valor de la consulta y monto pagado en esa consulta (puede ser 0 o parcial), actualizando automáticamente `pagado` y `pendiente`; b) abonar a la deuda de un paciente en cualquier momento; c) mostrar estadísticas incluyendo un listado de pacientes con deuda ordenado de mayor a menor pendiente; todo validando entradas con `InputMismatchException` y rangos correctos. Usa Programación Estructurada (PE) con tres clases: `Ejercicio120_Main` (coordina el flujo), `Ejercicio120_Utilidades` (maneja sello personalizado "Cristina Cruz | Lis ❤️ | cristinacruz@unah.hn | 120 | 2025-10-23", menús bilingües español/inglés, validación de entradas, presentación de estadísticas y consultas), y `Ejercicio120_Funciones` (inicializa arreglos, registra consultas, abona pagos, ordena deudores); el programa debe incluir menú de ejecución ("1. Ejecutar", "2. Finalizar") y submenú con opciones (registrar consulta, abonar deuda, mostrar estadísticas), usar constantes para mensajes, mostrar el sello, y propor Razonamiento más profundo down con secciones de análisis, diseño modular, resultados y conclusiones, siguiendo el estilo de programas previos sin comentarios en el

## Salida de la ejecución del programa:

```
Menú de ejecución:
1. Ejecutar
2. Finalizar

Opción: 1
Sistema de gestión de consultas

1. Registrar consulta
2. Abonar a deuda
3. Mostrar estadísticas
4. Salir

Opción: 1
Expediente del paciente: 101
Año de la consulta: 2002
Mes (1-12): 8
Día (1-31): 15
Valor de la consulta: 500
Monto pagado en esta consulta (0 si nada): 0
```

```
Monto pagado en esta consulta (0 si nada): 0
Consulta registrada exitosamente.
```

```
1. Registrar consulta
```

```
2. Abonar a deuda
```

```
3. Mostrar estadísticas
```

```
4. Salir
```

```
Opción: 2
```

```
Expediente del paciente: 101
```

```
Monto a abonar: 250
```

```
Abono registrado.
```

```
1. Registrar consulta
```

```
2. Abonar a deuda
```

```
3. Mostrar estadísticas
```

```
4. Salir
```

```
Opción: 3
```

```
Opción: 3
Listado de pacientes:
```

```
-----
Expediente Nombre Sexo Diagnóstico
101 Ana López F Obesidad
102 Carlos Méndez M Diabetes
103 María Rivera F Hipertensión
```

```
Deudores (mayor a menor deuda):
```

```
-----
Expediente Nombre Pagado Pendiente
101 Ana López 250.00 250.00
```

```
1. Registrar consulta
```

```
2. Abonar a deuda
```

```
3. Mostrar estadísticas
```

```
4. Salir
```

```
Opción:
```

## EJERCICIO#121

### Enunciado

Registro V2: Vamos bien, muy satisfecho con sus avances, vamos a hacer con el ejercicio “Registros” lo que hizo con la clínica para ir mejorando; es decir, agregar expedientes de pacientes, pero este expediente no solo guarda el diagnóstico del peso sino la especialidad que lo está atendiendo y si está o no interno en el hospital, si está interno en qué cama de qué habitación de qué piso.

Uno puede modificar celular, estatura, peso y ahora:

1. Si está interno darle de alta, eso significa borrar la información de cama, habitación, piso y especialidad.
2. Si no está interno entonces puede internarse y agregar para ello especialidad, piso, habitación y cama.

Ahora las estadísticas muestran pacientes internos por especialidad

### Análisis de Requerimientos:

- **Sello personalizado:** Solo **VERSIÓN CORTA** (Dany Noguera | dany.noguera@unah.hn | 121 | 2025-11-06) dentro de los cuatro archivos; se muestra en un cuadro elegante separado durante la ejecución.
- **Programa:**
  - Crear una instancia de Paciente.
  - Inicializar el paciente.
  - Leer los datos personales de entrada (peso en libras y estatura en cm).
  - Calcular el IMC mediante funciones:
    - librasAKg(): convierte peso de libras a kilogramos.
    - cmAMetros(): convierte estatura de centímetros a metros.
    - calcularIMC(): calcula IMC usando fórmula ( $IMC = kg / (m^2)$  ).
  - Función asignarDx(paciente) para asignar diagnóstico según IMC.
  - Funciones de internamiento:
    - internar(paciente): permite asignar especialidad, piso, habitación y cama.
    - darDeAlta(paciente): borra información de internamiento y marca como no interno.
    - Imprimir información del paciente mediante función.
  - Mantener menú de idioma (ES/EN) y opción iniciar/finalizar con bucle.
  - Mantener consistencia estética con bordes Unicode y formato uniforme con ejercicios anteriores.
- **Modularidad:** Cuatro archivos en paquete Ejercicio121:
  - Ejercicio121\_Main.java Control del flujo principal; bucle iniciar/finalizar y llamada a operaciones principales.

- Ejercicio121\_Menus.java Manejo de menús: selección de idioma y menú iniciar/finalizar.
- Ejercicio121\_Utilidades.java Funciones gráficas: encabezados, pies, sello corto, impresión tabular de Paciente.
- Ejercicio121\_Operaciones.java Funciones de lógica: lectura, conversión de unidades, cálculo de IMC, diagnóstico, internamiento.
- **Paquete:** Ejercicio121.

**Estética:** Uso de bordes Unicode (‗, †, ‖, =) para encabezados y pies.  
 Matrices con encabezados de sucursales y días alineadas en columnas de ancho fijo.  
 Sello corto mostrado antes del pie final.  
 Resultados y mensajes tabulados y uniformes con ejercicios previos.

- **Paradigma:** Programación estructurada.  
 Uso de funciones y métodos estáticos.  
 Modularización completa en 4 archivos.  
 Uso de registros/estructuras para almacenar información.  
 Flujo secuencial con validaciones, condicionales y bucles.
- **Documentación:** -Metadatos en cada archivo + comentarios Javadoc en métodos.

#### Flujo Principal (en Ejercicio121\_Main):

- main():
- Selección de idioma.
- Bucle principal: ejecutar o finalizar el programa.
- Llamada a Ejercicio121\_Operaciones.registroOperacion() si se decide iniciar.

#### Subrutinas (en Ejercicio121\_Menus):

- seleccionarIdioma(): El usuario elige el idioma ES / EN.
- menuInicioSalir(): Pregunta si desea ejecutar el programa o salir.
- obtenerIdioma(): Devuelve idioma actual.

#### Subrutinas (en Ejercicio121\_Utilidades):

- mostrarEncabezado(String titulo): imprime un encabezado decorativo con bordes Unicode.
- mostrarPie(String msg): imprime un pie de mensaje decorativo.
- mostrarSelloCorto(): imprime el sello corto del desarrollador antes del pie final.
- imprimirPaciente(Paciente p): imprime los datos de un paciente de forma tabular, mostrando internamiento si aplica.

### **Subrutinas (en Ejercicio121\_Operaciones):**

- registroOperacion():
- Inicializa paciente.
- Llama a leerDatosPaciente(Paciente p) para capturar datos.
- Convierte peso a kg (librasAKg()) y estatura a metros (cmAMetros()).
- Calcula IMC con calcularIMC().
- Asigna diagnóstico con asignarDx().
- Permite internar o dar de alta al paciente mediante bucle de opciones.
- Imprime información completa del paciente.
- Muestra sello corto y pie final con mensaje de proceso completado.
- leerDatosPaciente(Paciente p): captura nombre, apellido, celular, sexo, peso y estatura.
- librasAKg(double libras): retorna peso en kg.
- cmAMetros(double cm): retorna estatura en metros.
- calcularIMC(double kg, double metros): retorna IMC.
- asignarDx(Paciente p): asigna diagnóstico según IMC.
- darDeAlta(Paciente p): borra información de internamiento y marca como no interno.
- internar(Paciente p): solicita especialidad, piso, habitación y cama, y marca como interno.

### **PROMPT**

Agente, respetando el orden y estructura que venimos trabajando y las reglas que te he proporcionado. El siguiente ejercicio es el 121 y su enunciado es: Registro V2: Vamos bien, muy satisfecho con sus avances, vamos a hacer con el ejercicio “Registros” lo que hizo con la clínica para ir mejorando; es decir, agregar expedientes de pacientes, pero este expediente no solo guarda el diagnóstico del peso sino la especialidad que lo está atendiendo y si está o no interno en el hospital, si está interno en qué cama de qué habitación de qué piso. Uno puede modificar celular, estatura, peso y ahora: 1. Si está interno darle de alta, eso significa borrar la información de cama, habitación, piso y especialidad. 2. Si no está interno entonces puede internarse y agregar para ello especialidad, piso, habitación y cama. Ahora las estadísticas muestran pacientes internos por especialidad. Recuerda usar \* Ciclos, funciones, librerías propias, arreglos y registros \* pero antes te brindare una platilla para que me brindes el ejercicio adaptandolo al actual

### **PROMPT GENERADO**

¡Perfecto, Dany! 😊

Entonces vamos a generar los **4 archivos del Ejercicio 121** adaptando todo lo del 117 a la nueva lógica de **Pacientes con internamiento, especialidad y cama**. Mantendremos **modularidad, estética, menús y funciones separadas** exactamente igual que antes.

Aquí va la propuesta completa:

### Ejercicio121\_Main.java

```
java Copiar código

// 
// | Desarrollador: Dany Noguera (dany.noguera@unah.hn)
// | Proyecto: Ejercicio #121
// | Prompt:
// | "Registro V2: Gestión de pacientes con internamiento, especialidad, cama,
// | piso y habitación. Posibilidad de dar de (↓), internar, modificar datos,
// | y estadísticas de internos por especialidad."
```

+ Pregunta lo que quieras

0

|||

## Salida de la ejecución del programa:

```
Run - Guía_de_ejercicios
Run Ejercicio121_Main x
C:\Users\Usuario\.jdks\openjdk-25\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.2.3\lib\idea_rt.jar=63240" -Dfile.encoding=UTF-8
↑ ↓ : 
↑ ↓ SELECCIÓN DE IDIOMA / LANGUAGE SELECTION
↑ ↓ 1. Español
↑ ↓ 2. English
↑ ↓ Seleccione / Select: 1
↑ ↓ MENÚ PRINCIPAL
↑ ↓ 1. Iniciar programa
↑ ↓ 2. Finalizar
↑ ↓ Seleccione una opción: 1
↑ ↓ REGISTRO DE PACIENTES
↑ ↓ Ingrese primer nombre: Hola
↑ ↓ Ingrese primer apellido: Nose
↑ ↓ Ingrese celular: 1311333
↑ ↓ Ingrese sexo (0-M,1-F): 0
↑ ↓ Ingrese peso en libras: 12
↑ ↓ Ingrese estatura en cm: 178
↑ ↓ 1. Dar de alta
↑ ↓ 2. Internar
↑ ↓ 3. Salir
↑ ↓ Seleccione opción:
```

## **TABLA DE CUMPLIMIENTO**

<b>Integrante de grupo</b>	<b>Ejercicio asignado</b>	<b>Ejercicios realizados</b>
Cristina Cruz	25	25
Erinson Alvaréz	15	15
Kevin Lainez	27	27
Dany Tinoco	27	27
Jonatan Varela	27	27