|  |  |  |
| --- | --- | --- |
| **Nombre**: Viktor Raúl Hernández Vargas  Leonardo Cerda Rodriguez | | **Matrícula**: AL03003728  AL07024282 |
| **Nombre del curso:**  Programación orientada a objetos | **Nombre del profesor**:  HECTOR ANTONIO AGUILAR MOGOLLAN | |
| **Módulo**:  1 | **Actividad**: Actividad 3 | |
| **Fecha**: 07/02/2025 | **Equipo**: N/A | |
| **Bibliografía**: | | |

**Introducción**

Para esta actividad, realizaré un programa en Java que permita gestionar información de productos a través de la creación de una clase llamada Producto. En esta clase, definiré atributos privados como el código, tipo, costo e impuesto, lo que me llevará a implementar métodos de acceso (get) y modificación (set) para asegurar la correcta manipulación de los datos, respetando los principios de encapsulamiento de la programación orientada a objetos.

Con esto, aprenderé a desarrollar métodos funcionales dentro de una clase, como calcularPrecio, que me permitirá calcular el precio de venta de un producto sumando un porcentaje de utilidad al costo base y añadiendo el impuesto correspondiente. Este proceso me ayudará a comprender cómo aplicar operaciones matemáticas simples en el contexto de objetos y atributos.

Además, crearé dos objetos de la clase Producto en la clase principal del programa, donde solicitaré al usuario que ingrese los valores de los atributos. Para asegurar la robustez del programa, implementaré estructuras de control de excepciones mediante bloques try-catch, lo cual me permitirá captar y manejar posibles errores durante la entrada de datos.

A través del método estático compararProductos, podré comparar los precios de venta de dos productos distintos e identificar cuál tiene el precio más alto. Esto me llevará a entender cómo trabajar con métodos estáticos, pasar objetos como parámetros y realizar comparaciones entre ellos.

Con esta actividad, fortaleceré mis conocimientos sobre programación orientada a objetos, encapsulamiento, manejo de excepciones, uso de métodos estáticos y la interacción con el usuario en Java.

**Parte 1.**

**Clase Producto**

class Producto {

private String codigo;

private String tipo;

private double costo;

private double impuesto;

public Producto() {}

public String getCodigo() {

return codigo;

}

public String getTipo() {

return tipo;

}

public double getCosto() {

return costo;

}

public double getImpuesto() {

return impuesto;

}

public void setCodigo(String codigo) {

this.codigo = codigo;

}

public void setTipo(String tipo) {

this.tipo = tipo;

}

public void setCosto(double costo) {

this.costo = costo;

}

public void setImpuesto(double impuesto) {

this.impuesto = impuesto;

}

public void muestraProducto() {

System.out.println("Código: " + codigo);

System.out.println("Tipo: " + tipo);

System.out.println("Costo: $" + costo);

System.out.println("Impuesto: " + impuesto + "%");

}

public double calcularPrecio(double utilidad) {

double precioAntesImpuesto = costo + (costo \* utilidad / 100);

double precioFinal = precioAntesImpuesto + (precioAntesImpuesto \* impuesto / 100);

return precioFinal;

}

}

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Explicacion**

Lo que hicimos en la clase Producto fue definir cuatro atributos privados: codigo, tipo, costo e impuesto. Decidimos hacerlos privados para asegurarnos de que solo puedan ser accedidos y modificados a través de métodos específicos, evitando que otras partes del programa los alteren directamente.

Para permitir el acceso a estos atributos de forma segura, implementamos métodos get. Estos métodos devuelven el valor de cada atributo cuando se les llama, lo que nos permite obtener la información de un producto sin modificarlo. Creamos getCodigo() para obtener el código del producto, getTipo() para su tipo, getCosto() para conocer el costo y getImpuesto() para obtener el porcentaje de impuesto asignado.

También creamos los métodos set, que permiten modificar los valores de los atributos desde fuera de la clase. Si queremos asignarle un nuevo código a un producto, usamos setCodigo(String codigo), mientras que setTipo(String tipo) nos permite cambiar su categoría. Para actualizar valores numéricos, usamos setCosto(double costo) y setImpuesto(double impuesto), asegurándonos de que cualquier modificación se haga de manera controlada.

Después, para mostrar los datos de un producto de manera organizada, hicimos el método muestraProducto(), que imprime en pantalla la información del producto. Con este método, podemos visualizar su código, tipo, costo e impuesto de forma clara.

Finalmente, implementamos el método calcularPrecio(double utilidad), que nos permite obtener el precio final de un producto. Recibe un porcentaje de ganancia como parámetro, lo suma al costo base y luego le aplica el impuesto correspondiente. De esta manera, calculamos correctamente el precio de venta, considerando el costo, la utilidad y el impuesto.

Así, logramos que los datos del producto sean seguros, accesibles solo mediante métodos controlados, y que el precio de venta se calcule correctamente según los valores ingresados.

**Parte 2.**

**Código Main**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Producto producto1 = new Producto();

Producto producto2 = new Producto();

System.out.println("Ingrese los datos del primer producto:");

ingresarDatos(producto1, scanner);

System.out.println("\nIngrese los datos del segundo producto:");

ingresarDatos(producto2, scanner);

System.out.println("\nDatos del primer producto:");

producto1.muestraProducto();

System.out.println("\nDatos del segundo producto:");

producto2.muestraProducto();

System.out.println("\nComparación de productos:");

System.out.println(compararProductos(producto1, producto2));

scanner.close();

}

public static void ingresarDatos(Producto producto, Scanner scanner) {

System.out.print("Código: ");

producto.setCodigo(scanner.nextLine());

System.out.print("Tipo: ");

producto.setTipo(scanner.nextLine());

while (true) {

try {

System.out.print("Costo: ");

producto.setCosto(Double.parseDouble(scanner.nextLine()));

break;

} catch (NumberFormatException e) {

System.out.println("Error: Ingrese un valor numérico válido para el costo.");

}

}

while (true) {

try {

System.out.print("Impuesto (%): ");

producto.setImpuesto(Double.parseDouble(scanner.nextLine()));

break;

} catch (NumberFormatException e) {

System.out.println("Error: Ingrese un valor numérico válido para el impuesto.");

}

}

}

public static String compararProductos(Producto p1, Producto p2) {

double precio1 = p1.calcularPrecio(20);

double precio2 = p2.calcularPrecio(20);

if (precio1 > precio2) {

return "El producto con mayor precio es:\n" +

p1.getCodigo() + " - " + p1.getTipo() + " con un precio de $" + precio1;

} else if (precio2 > precio1) {

return "El producto con mayor precio es:\n" +

p2.getCodigo() + " - " + p2.getTipo() + " con un precio de $" + precio2;

} else {

return "Ambos productos tienen el mismo precio de venta.";

}

}

}

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Funcionamiento**

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

**Explicación**

En el código que implementamos, la clase Main tiene como objetivo crear dos objetos de la clase Producto y solicitar al usuario que ingrese los datos correspondientes para cada uno de ellos. Para esto, se crean dos instancias de Producto (producto1 y producto2) y se utiliza un método estático llamado ingresarDatos para pedir la información y asignarla a los atributos de cada objeto. Dentro de este método, se emplea un bloque try-catch para manejar posibles errores de entrada, como cuando el usuario ingresa datos incorrectos (por ejemplo, valores no numéricos cuando se esperan de tipo double). Si ocurre un error, el bloque catch lo captura y muestra un mensaje de error al usuario, pidiéndole que ingrese los datos correctamente.

La ejecución del programa comienza en el método main, donde se crea un objeto Scanner para la entrada de datos, se instancian dos objetos de la clase Producto (producto1 y producto2), y se llama al método ingresarDatos para solicitar los datos de cada producto al usuario. Luego, se muestran los detalles de los productos utilizando el método muestraProducto, y posteriormente se comparan los precios mediante el método compararProductos.

El método ingresarDatos recibe un objeto Producto y el Scanner como parámetros. Solicita al usuario ingresar el código y el tipo del producto, almacenándolos directamente. Para los valores numéricos (costo e impuesto), se emplea un bucle while con try-catch, de manera que si el usuario ingresa un valor no válido (por ejemplo, texto en lugar de un número), el programa muestra un mensaje de error y le pide que vuelva a intentarlo hasta que la entrada sea correcta.

El método compararProductos recibe dos objetos Producto, calcula el precio de venta de cada uno invocando el método calcularPrecio (que considera el costo, el impuesto y una utilidad fija del 20%), y determina cuál producto tiene el mayor precio. Luego, devuelve un mensaje con la información del producto más caro o indica si ambos tienen el mismo precio.

Finalmente, en el main, se invoca el método compararProductos y se muestra el resultado en pantalla. Para evitar problemas de memoria, el Scanner se cierra al final del programa. Este programa permite ingresar datos de manera segura, evitando errores de entrada, y compara los precios de los productos de forma clara y estructurada.

**Explicación de la clase y código fuente**

Ambos archivos (la clase Producto y la clase Main) interactúan de manera fluida para cumplir con el propósito de crear, mostrar y comparar productos. La clase Producto define los atributos y comportamientos de un producto. Esta clase tiene cuatro atributos privados: codigo, tipo, costo e impuesto, que representan las características principales del producto. Los métodos get y set permiten acceder y modificar estos atributos de manera controlada, asegurando que solo se cambien de forma válida. Además, la clase incluye el método muestraProducto, que imprime en pantalla la información completa del producto, y el método calcularPrecio, que calcula el precio final de venta, tomando en cuenta el costo base, un impuesto y una utilidad fija.

La clase Main se encarga de la interacción con el usuario. En el método main, se crean dos instancias de la clase Producto (producto1 y producto2) y se le solicita al usuario que ingrese los datos de cada uno de ellos. Esto se hace a través del método estático ingresarDatos, que recibe el objeto producto y el Scanner para obtener la entrada del usuario. Si el usuario introduce datos incorrectos, el bloque try-catch captura el error y le solicita al usuario que ingrese los datos correctamente. Una vez que los datos son ingresados correctamente, se llama al método muestraProducto para imprimir los detalles de ambos productos en pantalla. Luego, se invoca el método estático compararProductos, que recibe ambos productos, calcula su precio de venta usando el método calcularPrecio de la clase Producto, y compara cuál de los dos tiene el mayor precio. Dependiendo del resultado, devuelve un mensaje que informa cuál es el producto con el precio más alto o si ambos tienen el mismo precio.

La interacción entre estas clases ocurre de manera secuencial. Primero, el main crea los objetos producto1 y producto2 de la clase Producto. Luego, usa el método ingresarDatos para asignar valores a sus atributos a partir de la entrada del usuario. Posteriormente, se utilizan los métodos muestraProducto para mostrar los detalles de cada producto y compararProductos para comparar sus precios y mostrar cuál es más caro o si ambos son iguales. Este flujo de trabajo garantiza que los productos se creen y se comparen adecuadamente, y que el usuario reciba información clara sobre los detalles y precios de los productos que ha ingresado.

**Códigos juntos**

import java.util.Scanner;

class Producto {

private String codigo;

private String tipo;

private double costo;

private double impuesto;

public Producto() {}

public String getCodigo() { return codigo; }

public String getTipo() { return tipo; }

public double getCosto() { return costo; }

public double getImpuesto() { return impuesto; }

public void setCodigo(String codigo) { this.codigo = codigo; }

public void setTipo(String tipo) { this.tipo = tipo; }

public void setCosto(double costo) { this.costo = costo; }

public void setImpuesto(double impuesto) { this.impuesto = impuesto; }

public void muestraProducto() {

System.out.println("Código: " + codigo);

System.out.println("Tipo: " + tipo);

System.out.println("Costo: $" + costo);

System.out.println("Impuesto: " + impuesto + "%");

}

public double calcularPrecio(double utilidad) {

double precioAntesImpuesto = costo + (costo \* utilidad / 100);

return precioAntesImpuesto + (precioAntesImpuesto \* impuesto / 100);

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Producto producto1 = new Producto();

Producto producto2 = new Producto();

System.out.println("Ingrese los datos del primer producto:");

ingresarDatos(producto1, scanner);

System.out.println("\nIngrese los datos del segundo producto:");

ingresarDatos(producto2, scanner);

System.out.println("\nDatos del primer producto:");

producto1.muestraProducto();

System.out.println("\nDatos del segundo producto:");

producto2.muestraProducto();

System.out.println("\nComparación de productos:");

System.out.println(compararProductos(producto1, producto2));

scanner.close();

}

public static void ingresarDatos(Producto producto, Scanner scanner) {

System.out.print("Código: ");

producto.setCodigo(scanner.nextLine());

System.out.print("Tipo: ");

producto.setTipo(scanner.nextLine());

while (true) {

try {

System.out.print("Costo: ");

producto.setCosto(Double.parseDouble(scanner.nextLine()));

break;

} catch (NumberFormatException e) {

System.out.println("Error: Ingrese un valor numérico válido para el costo.");

}

}

while (true) {

try {

System.out.print("Impuesto (%): ");

producto.setImpuesto(Double.parseDouble(scanner.nextLine()));

break;

} catch (NumberFormatException e) {

System.out.println("Error: Ingrese un valor numérico válido para el impuesto.");

}

}

}

public static String compararProductos(Producto p1, Producto p2) {

double precio1 = p1.calcularPrecio(20);

double precio2 = p2.calcularPrecio(20);

if (precio1 > precio2) {

return "El producto con mayor precio es:\n" + p1.getCodigo() + " - " + p1.getTipo() + " con un precio de $" + precio1;

} else if (precio2 > precio1) {

return "El producto con mayor precio es:\n" + p2.getCodigo() + " - " + p2.getTipo() + " con un precio de $" + precio2;

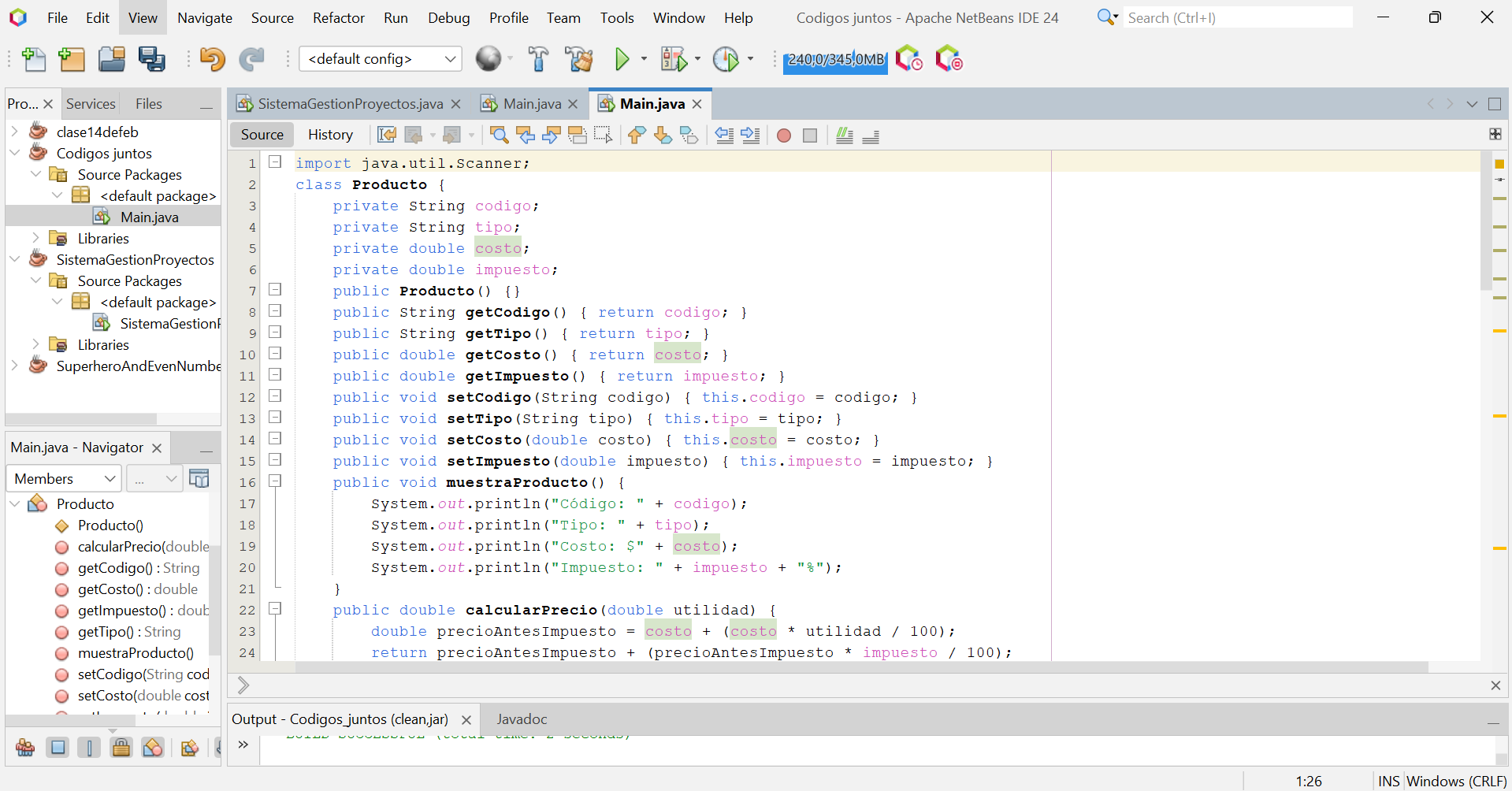
} else {

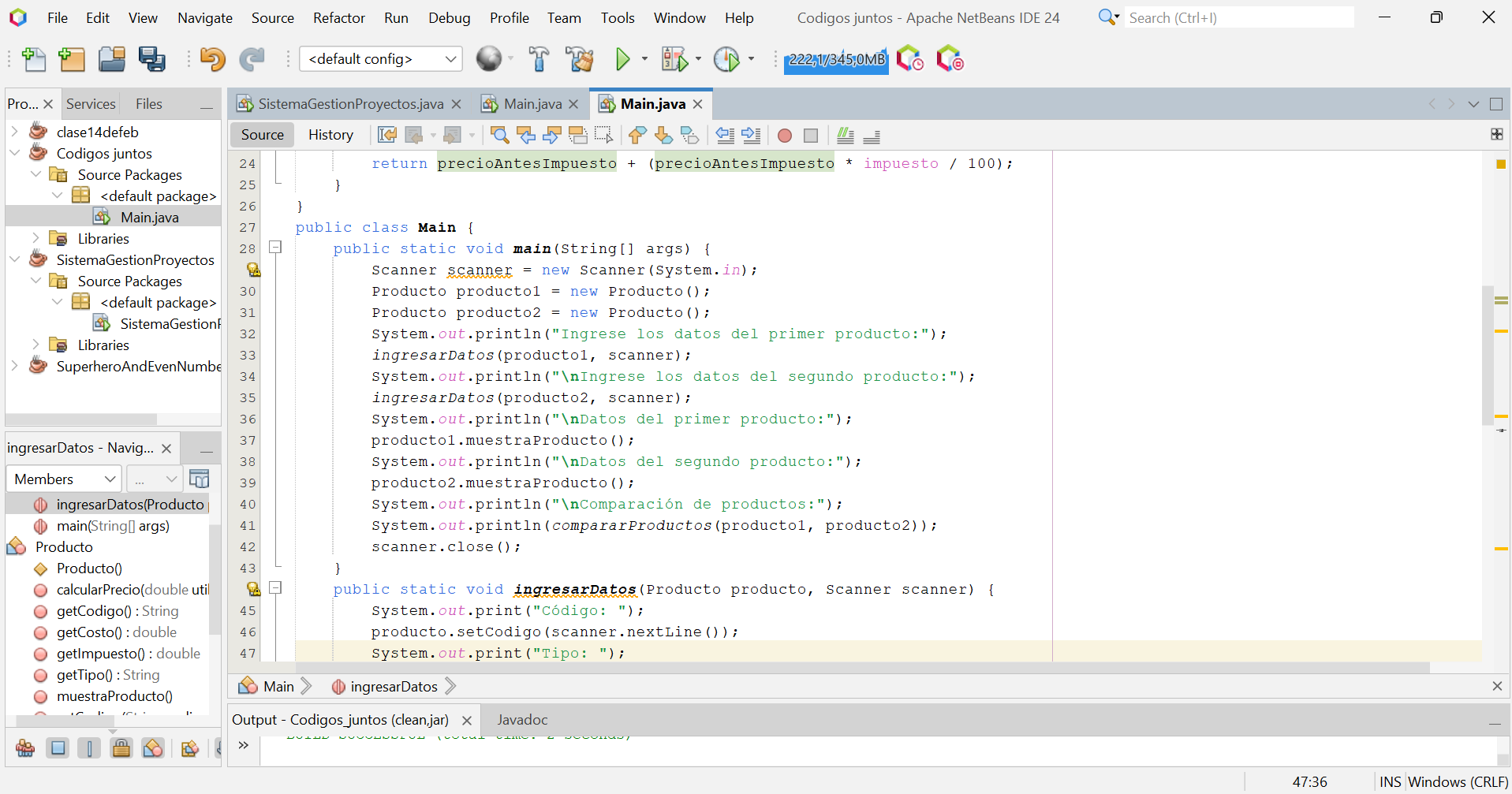
return "Ambos productos tienen el mismo precio de venta.";

}

}

}





Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Explicación**

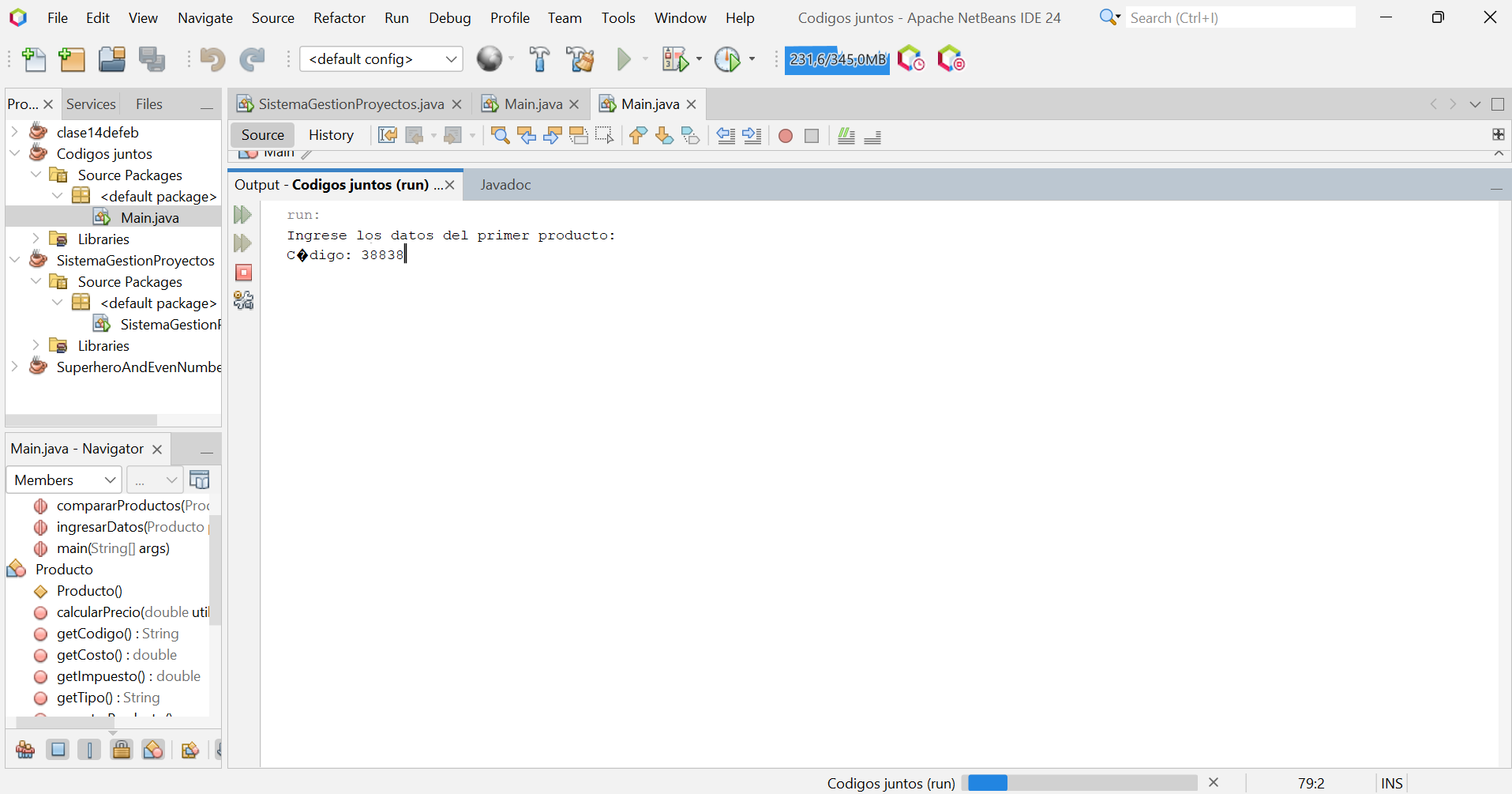
Lo que hicimos fue unir las dos partes del código en un solo archivo, manteniendo la estructura clara y funcional. Primero, tenemos la clase Producto, que define los atributos y métodos necesarios para gestionar productos con su código, tipo, costo e impuesto. También implementa métodos para obtener y modificar estos valores, mostrar la información del producto y calcular el precio final con una utilidad específica.

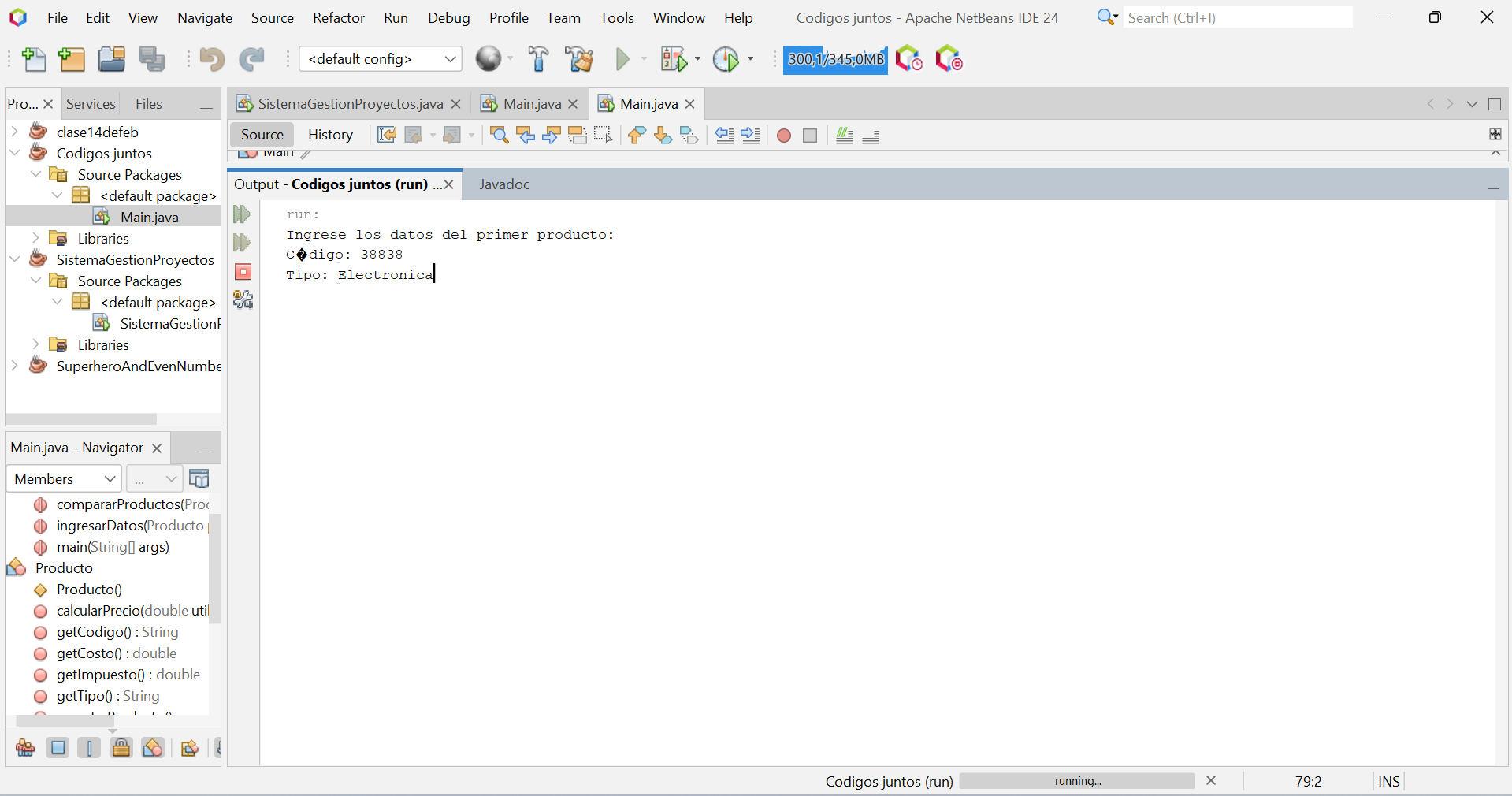
En la clase Main, que contiene el método main, manejamos las interacciones con el usuario. Creamos dos objetos de la clase Producto y solicitamos al usuario que ingrese sus datos mediante el método ingresarDatos. Este método asegura que los valores de costo e impuesto sean numéricos y maneja errores si el usuario ingresa un dato no válido.

Después de obtener los datos de ambos productos, los mostramos en pantalla usando el método muestraProducto, que simplemente imprime la información. Luego, comparamos los precios calculados con una utilidad del 20% mediante el método compararProductos. Esta comparación devuelve un mensaje indicando cuál de los dos productos tiene el precio de venta más alto o si ambos tienen el mismo precio.

La unión del código fue sencilla porque ya estaban bien organizadas las clases y los métodos. Simplemente juntamos la definición de la clase Producto antes de Main para que Main pudiera instanciar objetos de Producto sin problemas. Al hacer esto, aseguramos que la lógica de captura de datos, cálculo y comparación funcionara sin modificaciones adicionales.

**Funcionamiento**





Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

**Conclusión**

En esta actividad, desarrollamos una aplicación en Java aplicando los principios de la programación orientada a objetos para gestionar información de productos. Unimos dos partes del código en un solo archivo, organizando la estructura de manera clara y funcional.

Primero, implementamos la clase Producto, definiendo atributos privados como código, tipo, costo e impuesto, asegurando la encapsulación mediante métodos get y set. También añadimos el método muestraProducto para visualizar la información del producto y calcularPrecio, que permite obtener el precio de venta considerando un porcentaje de utilidad y el impuesto correspondiente.

Luego, en la clase Main, creamos dos objetos de la clase Producto y solicitamos al usuario que ingresara los valores de sus atributos mediante el método ingresarDatos. En este método, utilizamos try-catch para validar que los valores de costo e impuesto fueran numéricos, garantizando así la robustez del programa y evitando errores por entradas incorrectas. Posteriormente, mostramos en pantalla los datos de cada producto con muestraProducto.

Finalmente, implementamos el método compararProductos, que permitió comparar los precios de venta de ambos productos aplicando una utilidad del 20%, determinando cuál tenía el mayor precio o si eran iguales. Esta funcionalidad nos permitió reforzar el uso de métodos estáticos y la comparación de objetos en Java.

A través de esta actividad, consolidamos conceptos clave de la programación orientada a objetos, como la encapsulación, el uso de métodos de acceso y modificación, la gestión de excepciones, el trabajo con métodos estáticos y la manipulación de objetos en Java. Además, la integración de ambas partes del código nos ayudó a comprender cómo estructurar un programa de manera eficiente y modular para mejorar su mantenimiento y escalabilidad.

**Link a repositorio github**