

**Explorando los conceptos fundamentales de la POO**

Milagros Pozzo Fasini

2º Profesorado de Informática Semipresencial, CERP SW Colonia

Programación II

Prof. Domingo Pérez González

2025

Contenido

[**Introducción** 3](#_Toc194752076)

[**Tarea: Explorando Los Conceptos Fundamentales De La POO** 4](#_Toc194752077)

[**1-** **Enlace A Repositorio Personal Con El Proyecto Modificado:** 4](#_Toc194752078)

[**2-** **Documenta detalladamente:** 4](#_Toc194752079)

[ *¿Qué Modificaciones Realizaste En El Proyecto?* 4](#_Toc194752080)

[*Corrección de Código Ejecutable Fuera de un Método o Constructor* 8](#_Toc194752081)

[*Ejemplo de Clases y Objetos: Ejes estructurales de la POO* 9](#_Toc194752082)

[*Corrección de la Implementación de Métodos con @Override* 10](#_Toc194752083)

[*Corrección de la Implementación de List* 12](#_Toc194752084)

[*Manejo de Errores con el Archivo "preguntas.data*" 12](#_Toc194752085)

[*Ajustes en la Estructura del Proyecto* 13](#_Toc194752086)

[*Instalación y Configuración de Maven* 15](#_Toc194752087)

[*Corrección de Advertencias en la Documentación Javadoc* 18](#_Toc194752088)

[ *¿Qué Unidad Del Curso Seleccionaste Para Crear Las Preguntas?* 19](#_Toc194752089)

[ *Capturas De Pantalla Propias Mostrando El Funcionamiento De Tu Versión Del Proyecto.* 19](#_Toc194752090)

[**1. Modificación De La Clase Preguntas.java** 24](#_Toc194752091)

[**3. Manejo De Rutas Y Acceso A Archivos** 25](#_Toc194752092)

[**4. Errores Con Maven Y Configuración Del Entorno** 26](#_Toc194752093)

[**5. Documentación Y Buenas Prácticas** 26](#_Toc194752094)

[Conclusión 27](#_Toc194752095)

[**Referencias bibliográficas** 28](#_Toc194752096)

# **Introducción**

Este documento fue realizado en el marco del segundo año del Profesorado de Informática Semipresencial del CERP del Suroeste, para la asignatura Programación II.

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en el uso de "objetos", los cuales son instancias de "clases". En este paradigma, las clases se utilizan para modelar entidades del mundo real o conceptos abstractos. Una clase describe un tipo de objeto, incluyendo tanto la información (atributos o campos) como las operaciones (métodos) que el objeto puede realizar (Pressman, 2010, p. 141).

En este proyecto se utilizó Java, un lenguaje que promueve el uso de la POO desde su estructura básica. La idea de encapsulamiento, es el proceso de ocultar los detalles internos del funcionamiento de un objeto y sólo permitir el acceso a través de una interfaz bien definida (Umbaugh, et.al., 2005). Según Booch (2007), la encapsulación permite ocultar la implementación interna de los objetos. En el proyecto **PreguntasDelCursoMVC**, esto se evidencia en el uso de clases que tienen atributos privados y métodos públicos para acceder a ellos (getters y setters).

Asimismo, se hace presente el principio de la reutilización de código a través de la herencia: La herencia permite a los programadores definir una nueva clase que reutiliza, extiende y modifica el comportamiento definido en otra clase (Eckel,2009, p. 8). En este proyecto, si bien no se observa herencia en todas las clases, la estructura modular permitiría fácilmente heredar de una clase base para extender funcionalidades.

Otro concepto importante es el polimorfismo, definido como la habilidad de una variable, función o objeto de tomar múltiples formas (Pressman, 2010, p. 747). Este concepto es útil en la arquitectura del patrón MVC (Modelo-Vista-Controlador), ya que permite diseñar componentes que puedan comportarse de diferentes maneras según su contexto de ejecución.

Finalmente, la programación modular facilita la organización del código. Dividir el código en módulos ayuda a los programadores a escribir código más claro, más fácil de depurar y mantener (Pino, et.al, 2021, p. 106).

# **Tarea: Explorando Los Conceptos Fundamentales De La POO**

## **Enlace A Repositorio Personal Con El Proyecto Modificado:**

<https://github.com/MilagrosPozzo/PreguntasDelCursoMVC>

## **Documenta detalladamente:**

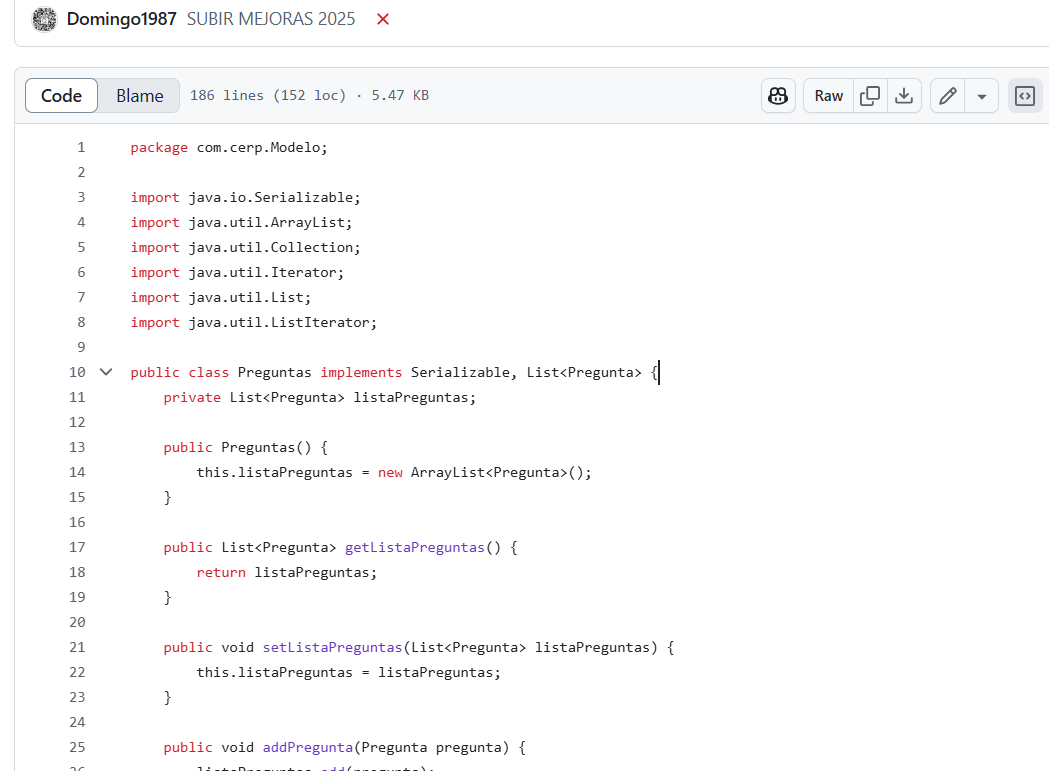
### *¿Qué Modificaciones Realizaste En El Proyecto?*

Las preguntas fueron agregadas en **Preguntas.java**, dentro del paquete **com.cerp.Modelo**, que es la clase encargada de manejar las preguntas.[[1]](#footnote-1)

Antes, la lista de preguntas (**listaPreguntas**) se inicializaba vacía en el constructor, sin preguntas predefinidas, por lo que solo contenía preguntas si se agregaban manualmente mediante métodos específicos.

**Figura 1**

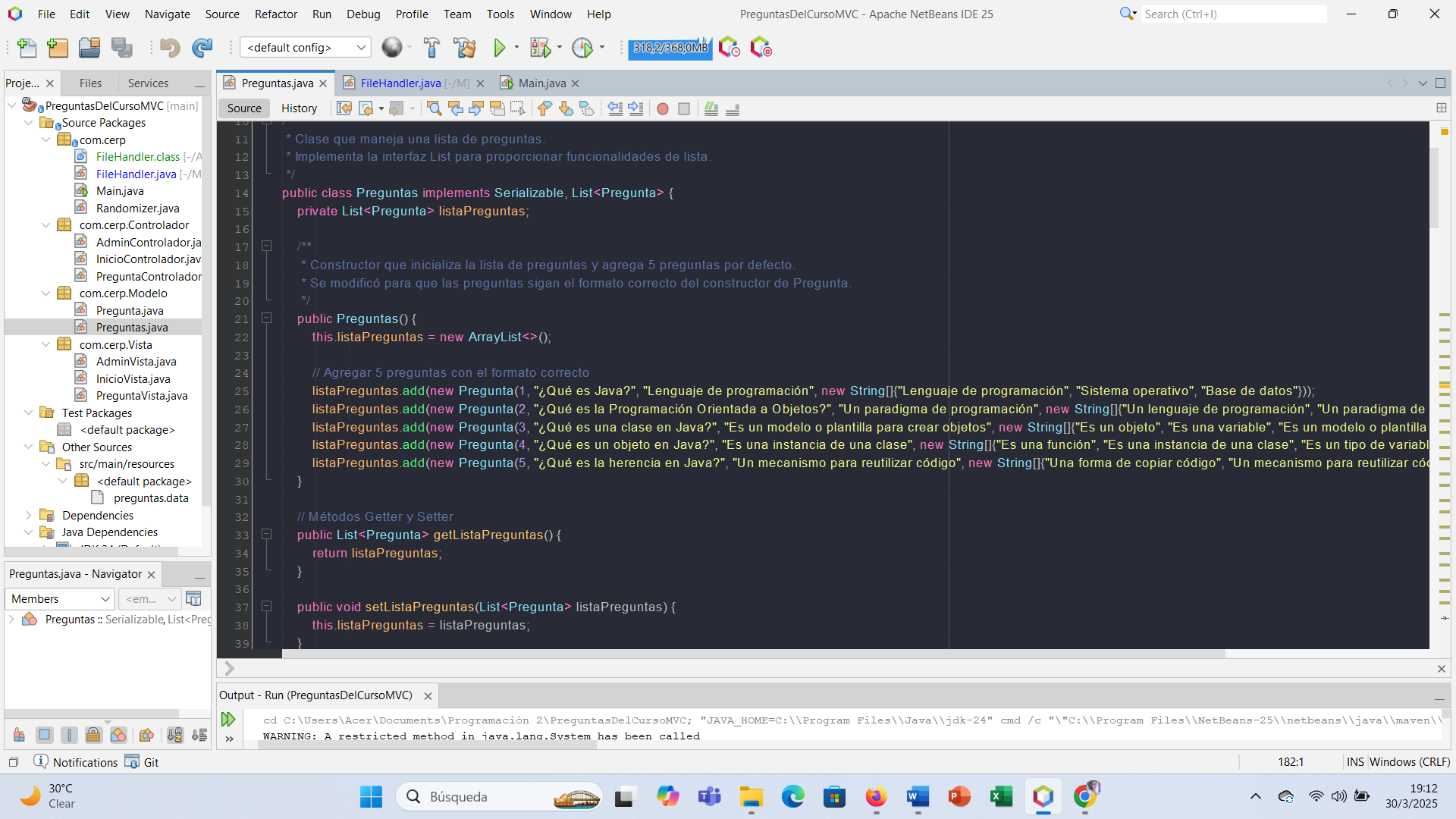
*Fragmento de código Java de la clase Preguntas dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración de la clase, la importación de librerías, la declaración de la lista de preguntas y el constructor.*



*Nota.* Elaboración propia.

Posteriormente, se modificó el código para incluir cinco preguntas dentro del constructor de **Preguntas.java**, asegurando que sigan el formato correcto del constructor de **Pregunta**. Estas preguntas incluyen el enunciado, la respuesta correcta y una lista de opciones. Se garantizó que los objetos de **Pregunta** creados dentro del constructor cumplan con la estructura adecuada, evitando problemas de inicialización.

**Figura 2**

*Fragmento de código Java de la clase Preguntas dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración de la clase, la importación de librerías, la inicialización de la lista de preguntas y la adición de preguntas predefinidas en el constructor.*

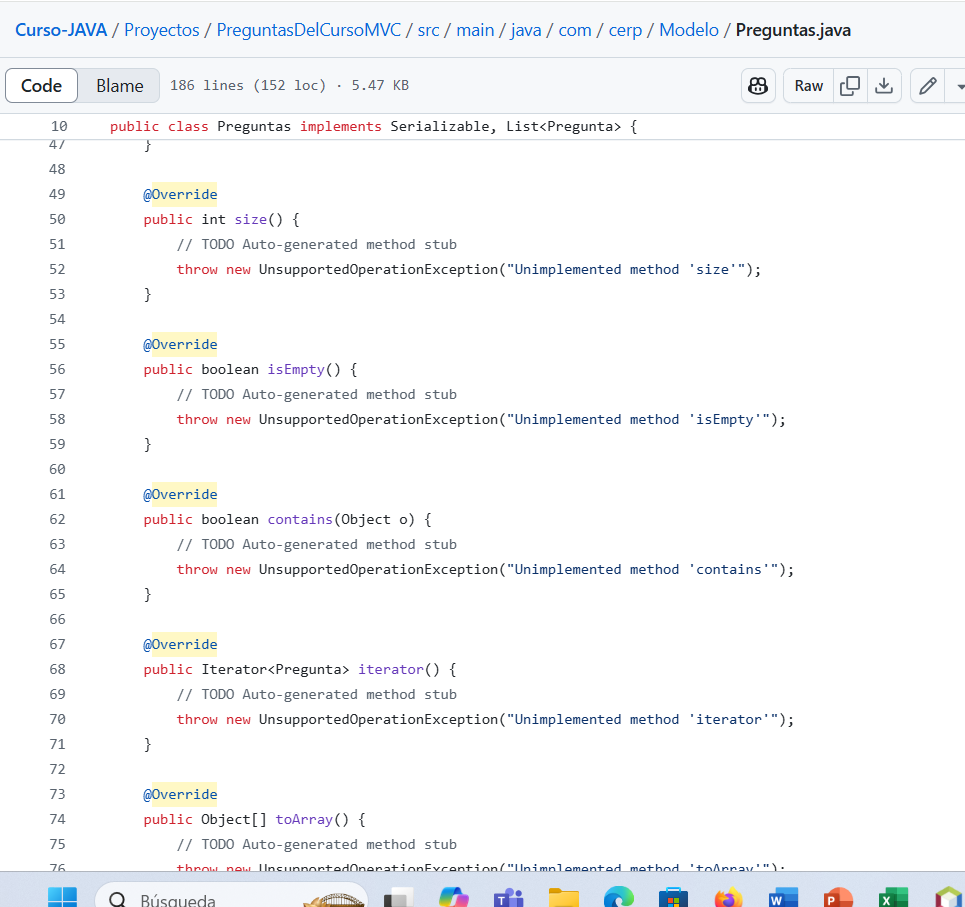
*Nota.* Elaboración propia.

Se verificó que la estructura del código respete la serialización y la gestión de la lista de preguntas. Se probaron los cambios ejecutando la aplicación para confirmar que las preguntas se cargan correctamente y que la implementación de los métodos de **List** funciona adecuadamente.

Antes, muchos métodos de la interfaz **List** lanzaban excepciones **UnsupportedOperationException.**

**Figura 3**

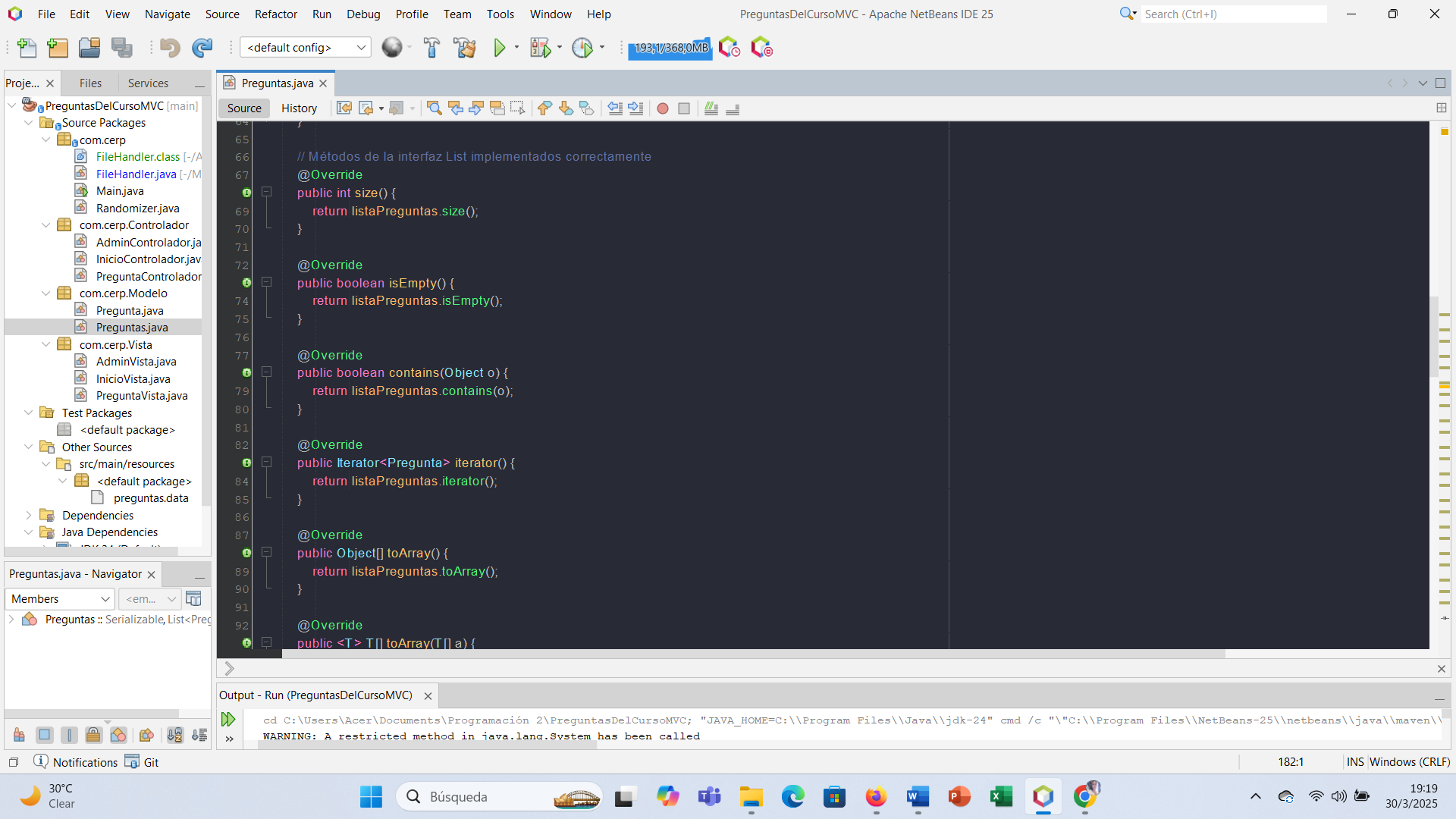
*Fragmento de código Java de la clase Preguntas dentro del proyecto PreguntasDelCursoMVC, mostrando la implementación de varios métodos de la interfaz List (size(), isEmpty(), contains(), iterator(), toArray()) con la indicación de que aún no han sido implementados (throw new UnsupportedOperationException).*



*Nota.* Elaboración propia.

Ahora, estos métodos fueron implementados correctamente, delegando su funcionalidad a **listaPreguntas:**

**Figura 4**

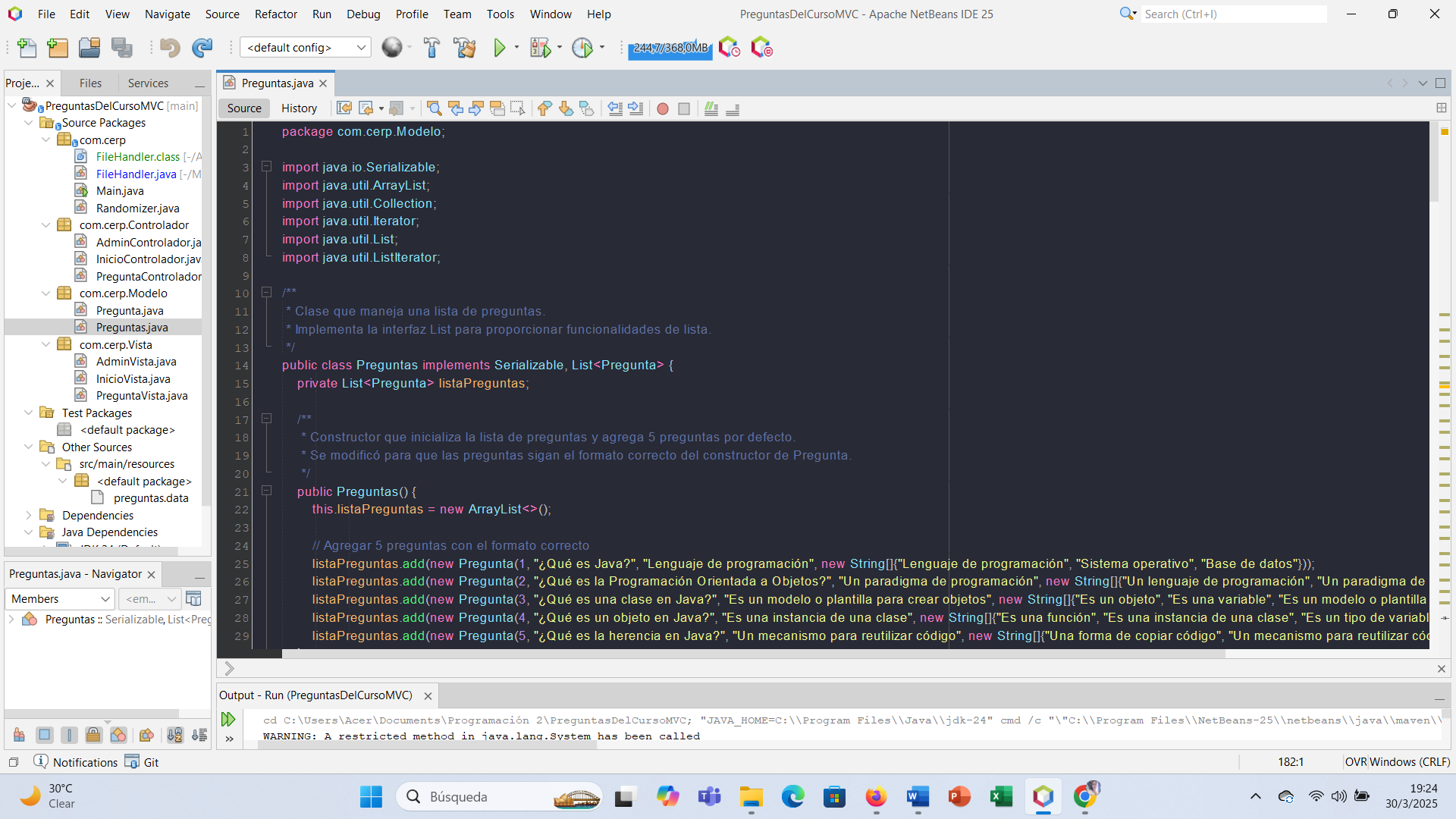
*Fragmento de código Java de la clase Preguntas dentro del proyecto PreguntasDelCursoMVC, mostrando la implementación de varios métodos como size(), isEmpty(), contains(), iterator() y toArray(), todos anotados con @Override.*

*Nota.* Elaboración propia.

Se añadieron comentarios explicativos para mejorar la comprensión del código:

**Figura 5**

*Fragmento de código Java de la clase Preguntas dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración de la clase, las importaciones, la inicialización de la lista de preguntas y la adición de preguntas predefinidas en el constructor.*

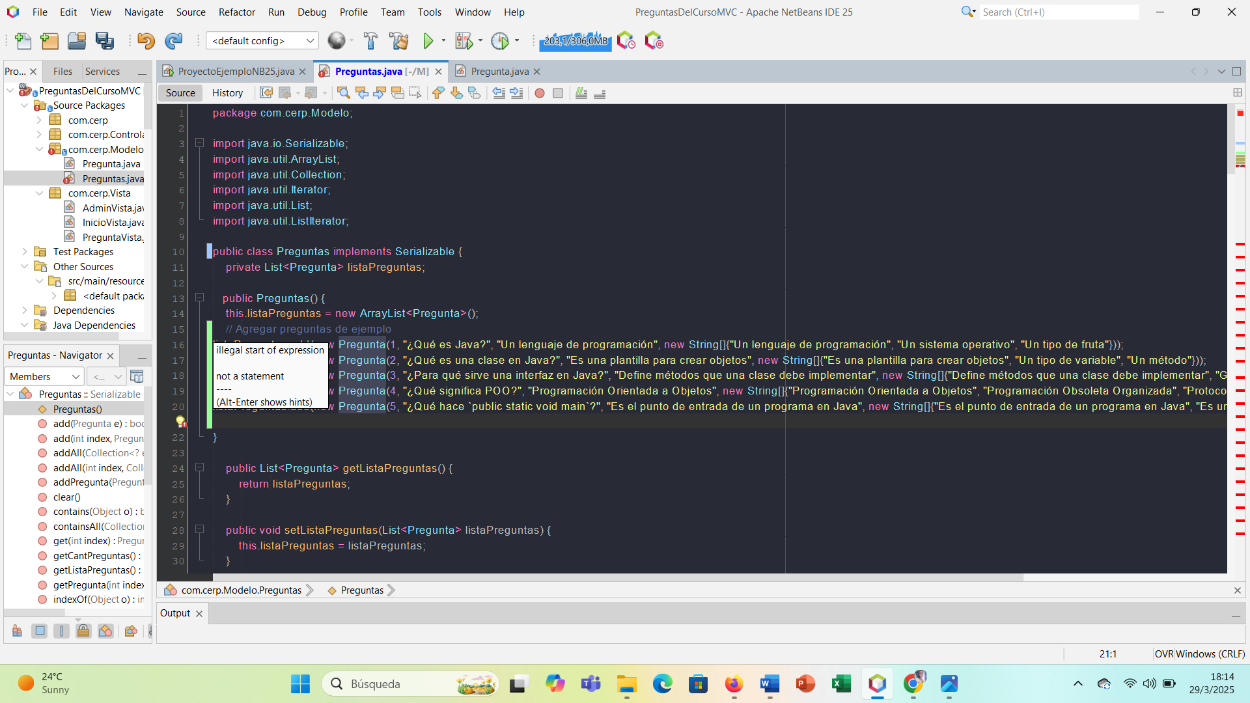


*Nota.* Elaboración propia.

### *Corrección de Código Ejecutable Fuera de un Método o Constructor*

En **Preguntas.java**, se detectó código ejecutable fuera de un método o constructor. Para corregirlo, todas las llamadas a métodos que agregan preguntas a la lista fueron ubicadas dentro del constructor **Preguntas()**.

**Figura 6**

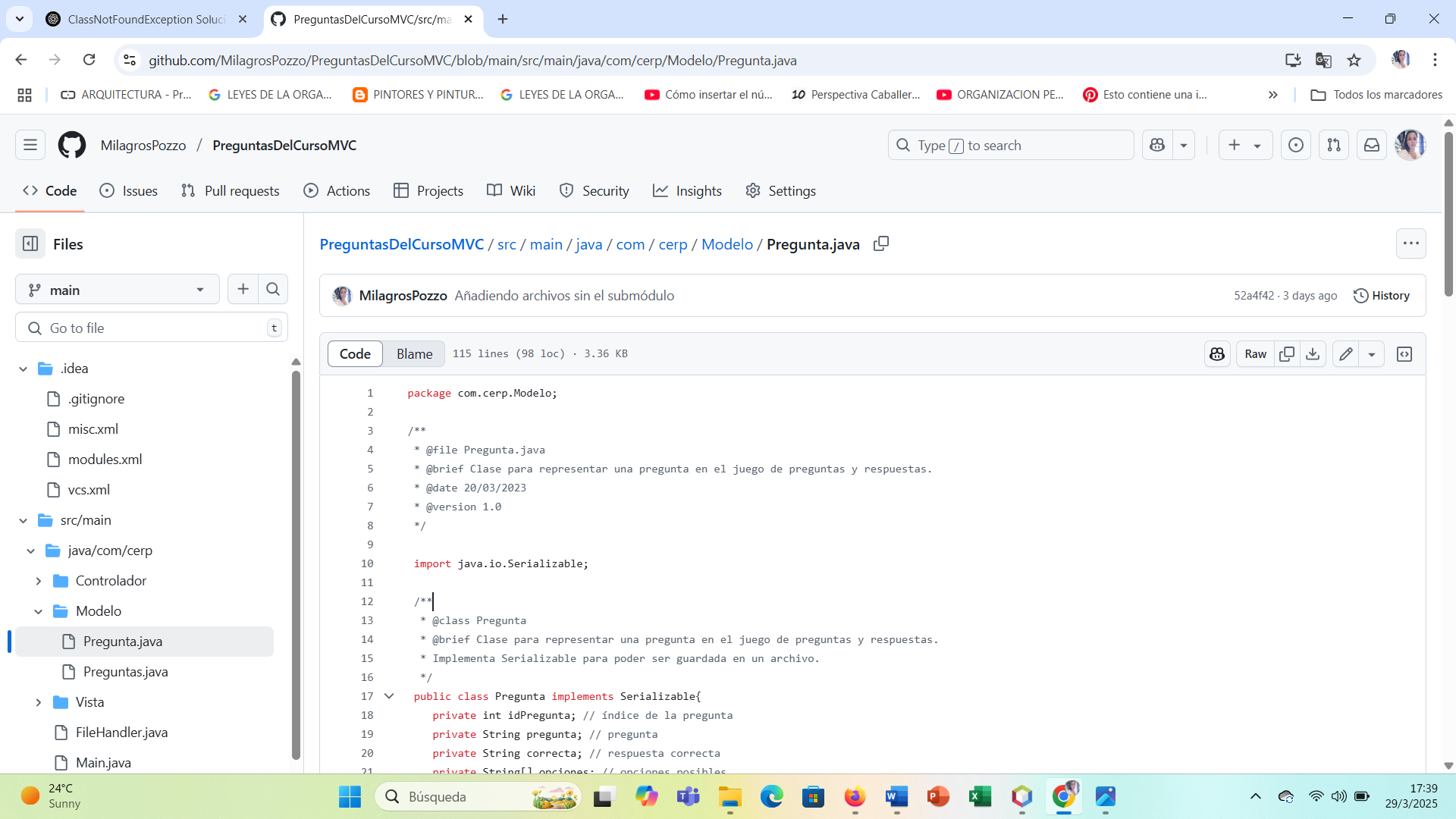
*Fragmento de código Java de la clase Preguntas dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración de la clase, la importación de librerías y la inicialización de la lista de preguntas.*

*Nota.* Elaboración propia.

El archivo **Pregunta.java** se encuentra dentro del **paquete Modelo**, que forma parte de la estructura **MVC** del **proyecto cerp**. Este archivo contiene la última versión confirmada de la clase **Pregunta**:

**Figura 7**

*Captura de pantalla del archivo Pregunta.java ubicado en la ruta src/main/java/com/cerp/Modelo/ dentro del proyecto PreguntasDelCursoMVC en GitHub.*

****

*Nota.* Elaboración propia.

Además, se detectó un error en la creación de objetos **Pregunta**, ya que se intentaba instanciarlos con solo dos parámetros (**String, String**), cuando el constructor de **Pregunta** espera **cuatro parámetros** (**int, String, String, String[]**).

**Para solucionar este problema:**

1. **Se movieron todas las instrucciones para agregar preguntas dentro del constructor Preguntas().**
2. **Se mantuvieron los métodos getter y setter para la lista de preguntas.**
3. **Se importaron las clases necesarias.}**

### *Ejemplo de Clases y Objetos: Ejes estructurales de la POO*

En la POO, las clases actúan como moldes o plantillas que definen atributos (características) y métodos (comportamientos) comunes a un conjunto de objetos. Los objetos, por su parte, son instancias concretas de esas clases. Por ejemplo, una clase “Vehículo” puede tener atributos como “color” o “velocidad”, y métodos como “acelerar” o “frenar”. Un objeto sería un “Auto rojo que acelera a 100 km/h”.

Este enfoque refleja cómo las personas categorizamos y organizamos la información, lo que hace que la POO sea especialmente intuitiva para estudiantes al modelar problemas reales mediante estructuras lógicas.

***Encapsulamiento, herencia y polimorfismo***

El encapsulamiento permite proteger la información sensible de un objeto, restringiendo el acceso a ciertos atributos y métodos. Esta práctica no solo mejora la seguridad del software, sino que promueve la claridad del código.

La herencia es otro concepto clave, que permite a una clase (subclase) heredar atributos y métodos de otra (superclase), favoreciendo la reutilización del código. Por ejemplo, una clase “Auto” puede heredar de la clase “Vehículo”, agregando propiedades propias como “número de puertas”.

El polimorfismo, por su parte, permite que diferentes clases respondan de forma distinta al mismo método. Esto facilita la abstracción y la flexibilidad en el diseño del software.

### *Corrección de la Implementación de Métodos con @Override*

Durante el desarrollo, NetBeans mostró advertencias sobre la falta de documentación en FileHandler.java y sobre el uso incorrecto de la anotación @Override.

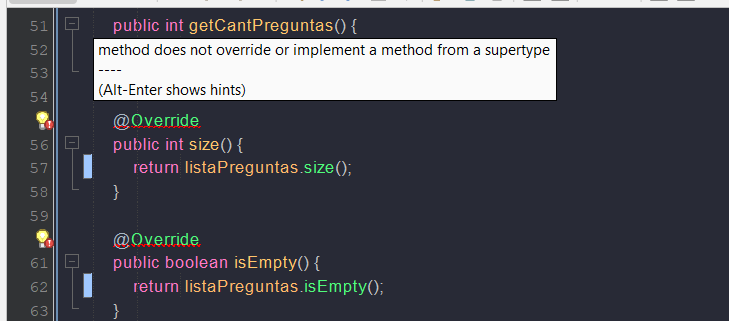
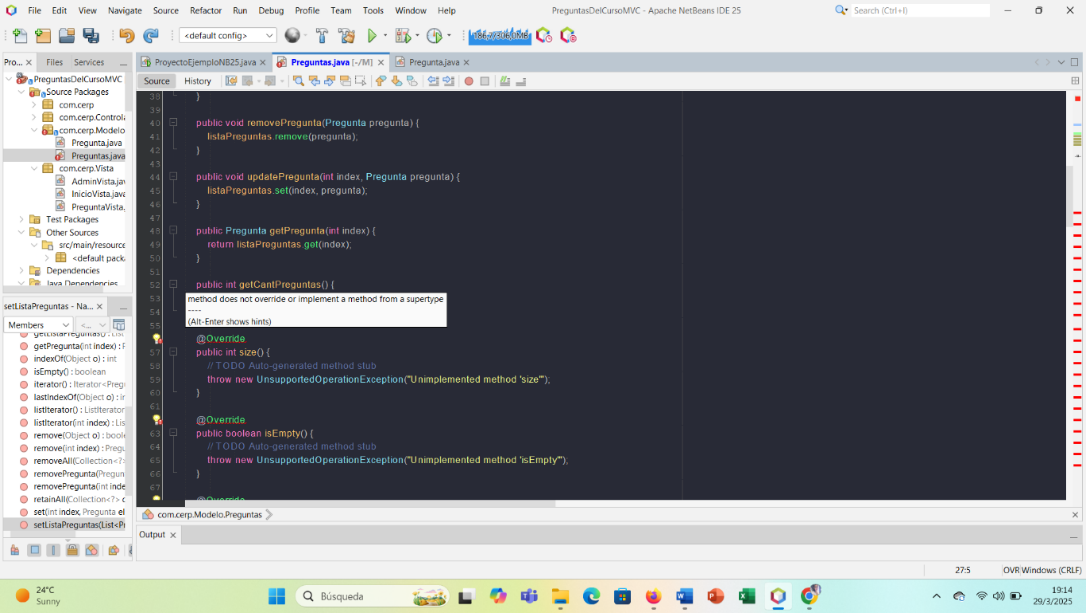
Dificultad: El código no tenía comentarios explicativos, lo que dificultaba su comprensión. Además, la anotación @Override se utilizaba en métodos que no sobrescribían ninguna superclase.

Solución: Se agregaron comentarios Javadoc detallados para explicar el propósito de cada clase y método. También se corrigieron las anotaciones @Override para aplicarlas únicamente en métodos que realmente sobrescribían otros de una superclase o interfaz.

El mensaje de error **"method does not override or implement a method from a supertype"** indicaba que se estaba utilizando la anotación **@Override** en métodos que no sobrescribían ningún método de una superclase o interfaz.

**Figura 8**

*Fragmento de código Java mostrando la implementación de los métodos size() e isEmpty() con la anotación @Override*

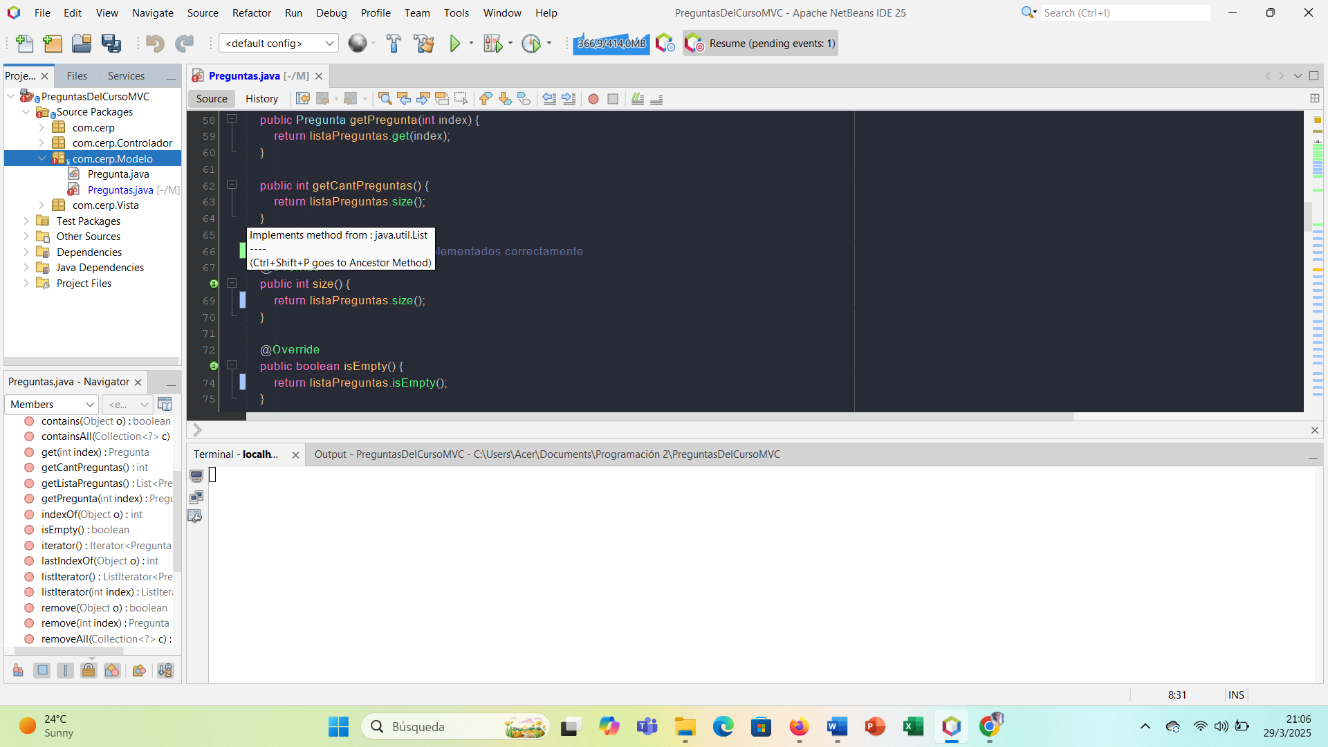


*Nota.* Elaboración propia.

Para corregirlo:

1. Se implementó **size()** para que devuelva el tamaño real de **listaPreguntas**.
2. Se implementó **isEmpty()** para verificar si la lista está vacía.
3. Se revisó **getCantPreguntas()** para corregir su implementación o eliminar la anotación **@Override** si no sobrescribía un método.

**Figura 9**

*Captura de pantalla del entorno de desarrollo integrado Apache NetBeans mostrando el código Java de la clase Preguntas.java dentro del proyecto PreguntasDelCursoMVC*.

*Nota.* Elaboración propia.

### *Corrección de la Implementación de List*

La clase **Preguntas** no debería implementar **List<Pregunta>**, ya que no sobrescribía todos los métodos de la interfaz correctamente. Se corrigió eliminando **List<Pregunta>** de la declaración de la clase (**línea 7).**

También se agregó la inicialización de preguntas de ejemplo dentro del constructor, en la línea 11.

### *Manejo de Errores con el Archivo "preguntas.data*"

El programa no encontraba el archivo **preguntas.data** debido a una referencia incorrecta en **FileHandler.java**. El archivo se encontraba en **src/main/resources/**, pero el código intentaba acceder a él desde la raíz del proyecto.

Para solucionar este problema:

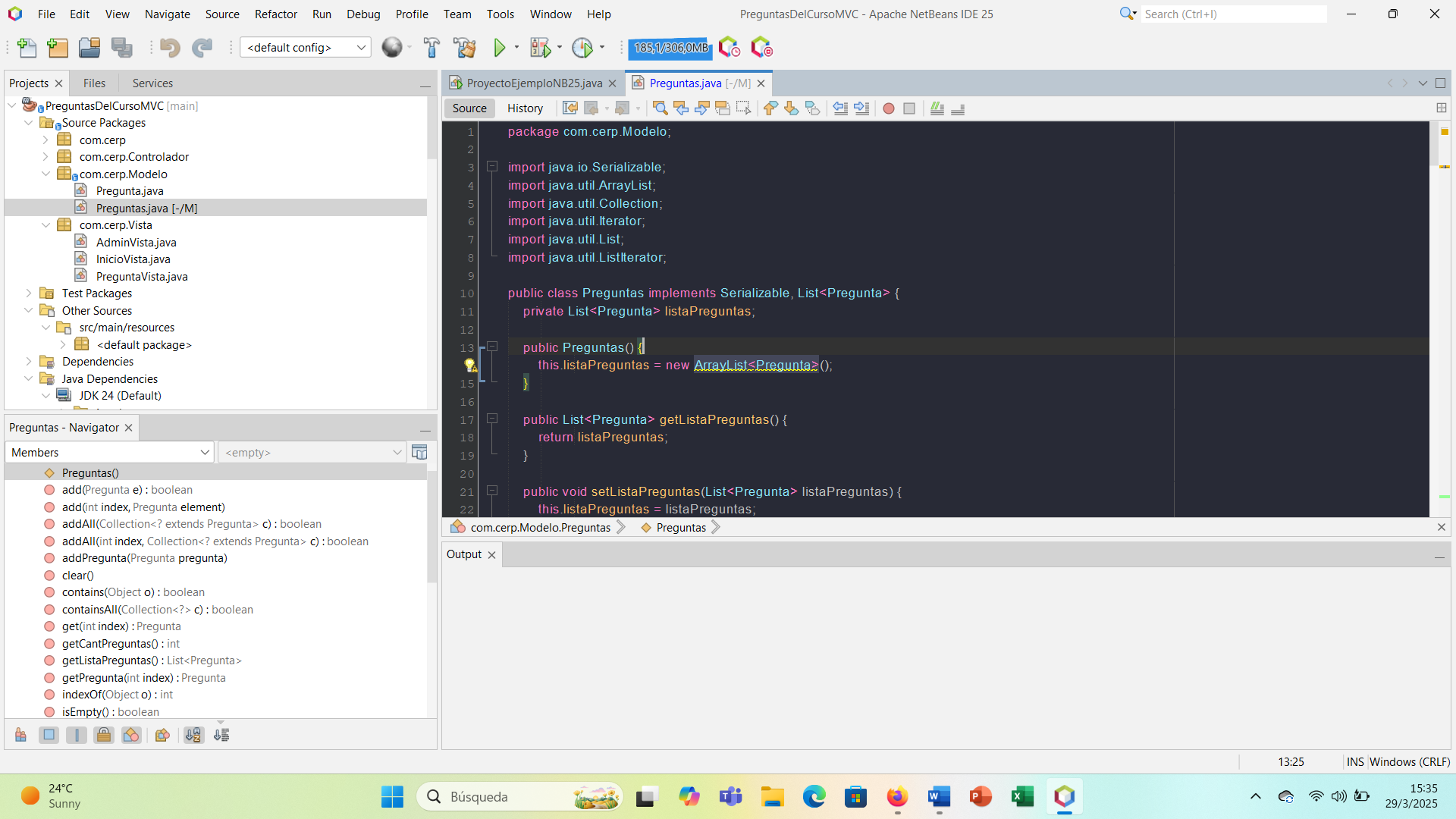
1. Se actualizó la ruta del archivo en el método **FileHandler.fileToList()** para usar una ruta relativa.
2. Se utilizó un cargador de clases para acceder al archivo dentro del directorio de recursos:

**InputStream inputStream = getClass().getClassLoader().getResourceAsStream("preguntas.data");**

1. Se modificó **Main.java** para utilizar preguntas por defecto si **preguntas.data** estaba vacío.
2. Se agregó código en **Main.java** para verificar si **preguntas.data** no existe o está vacío, en cuyo caso se carga la lista de preguntas desde **Preguntas.java**.

**Figura 10**

*Fragmento de código Java de la clase Preguntas dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración de la clase, la importación de librerías, la declaración de la lista de preguntas y la inicialización de la lista en el constructor.*



*Nota.* Elaboración propia

### *Ajustes en la Estructura del Proyecto*

En NetBeans, para acceder a **FileHandler.java**:

Abrir la carpeta "**Source Packages**".

Dentro de esta, abrir **com.cerp**.

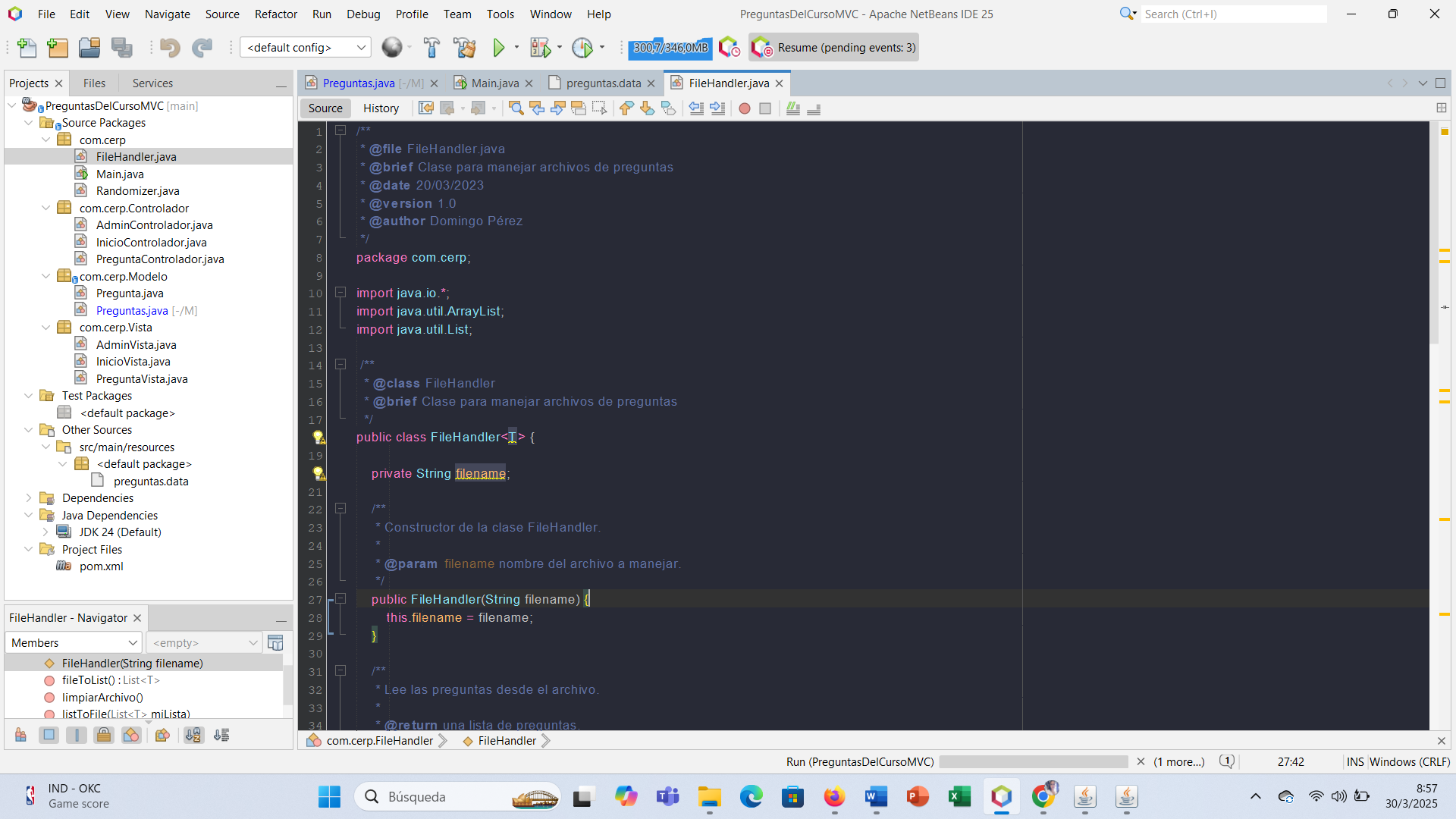
Allí se encuentra **FileHandler.java**.

Se modificó el constructor de FileHandler para manejar rutas correctamente: **Se agregó la línea 9**:

**Se agregó el siguiente bloque de código dentro del try (línea 40)**:**Para que el juego use Preguntas.java y**  no **preguntas.data**:

**Figura 11**

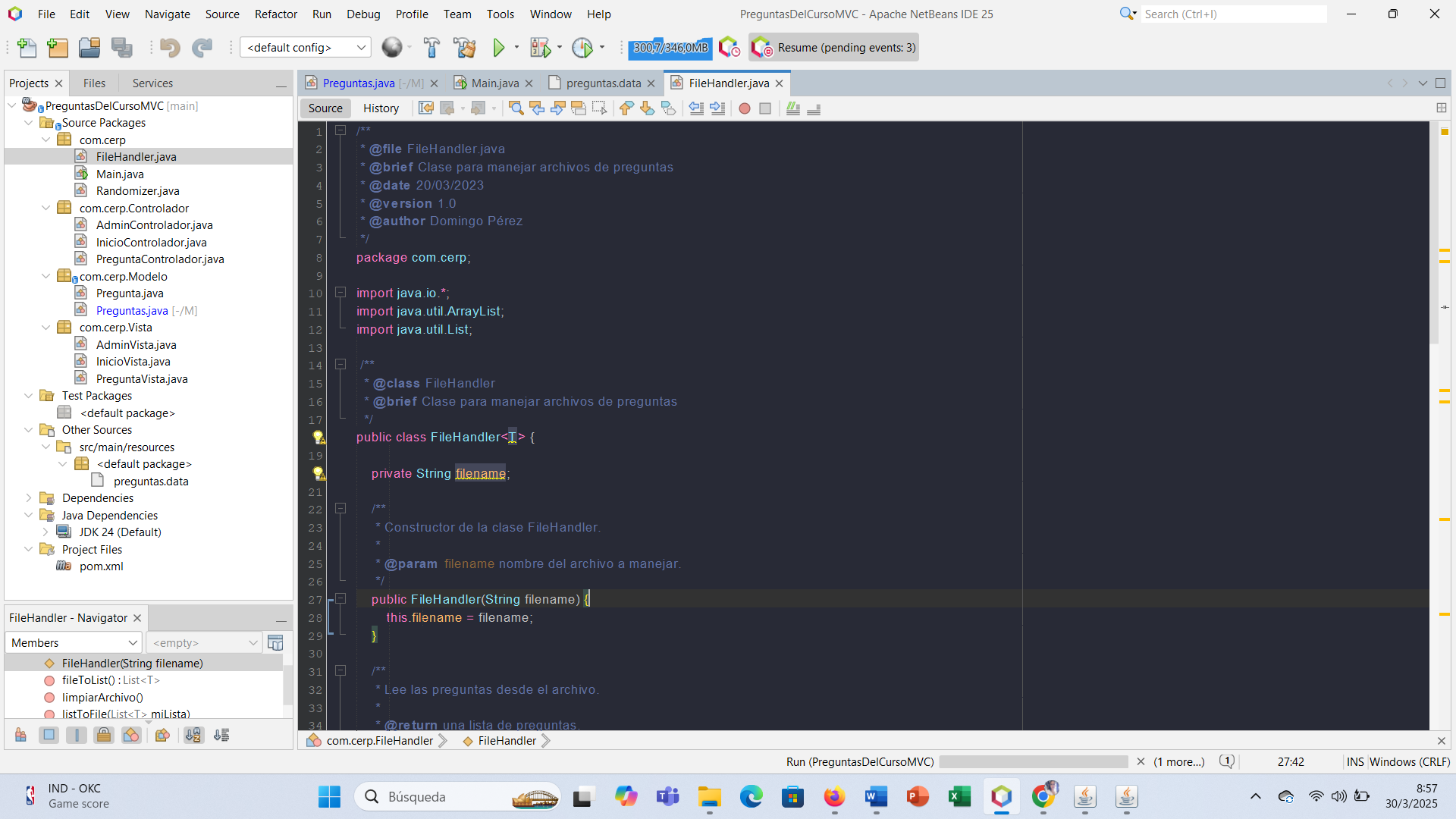
*Fragmento de código Java de la clase FileHandler dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración de la clase, la importación de librerías y el constructor que recibe el nombre del archivo*.



*Nota.* Elaboración propia

**Figura 12**

*Fragmento de código Java de la clase FileHandler dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración de la clase, la importación de librerías y la definición del constructor que recibe el nombre del archivo a manejar.*



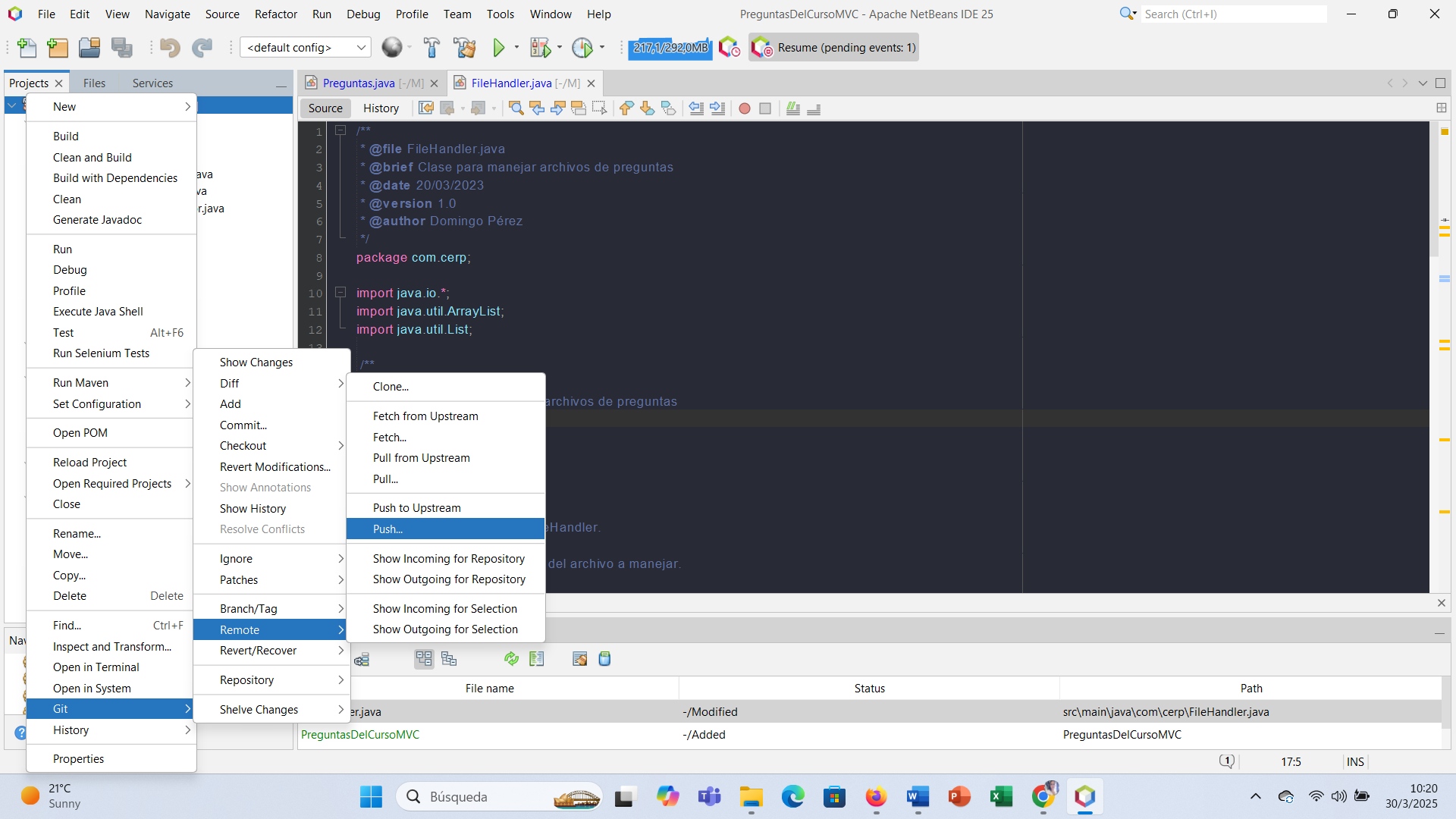
*Nota.* Elaboración propia.

**Limpiar y reconstruir** el proyecto en NetBeans:

**Run > Clean and Build Project**

**Figura 13**

*Captura de pantalla del entorno de desarrollo integrado Apache NetBeans mostrando el menú contextual de Git para el archivo FileHandler.java dentro del proyecto PreguntasDelCursoMVC.*



*Nota.* Elaboración propia

Para subir los cambios a Github

**Git > Remote > Push...**: Esta es la opción principal que necesitas para enviar tus cambios locales a un repositorio remoto en GitHub

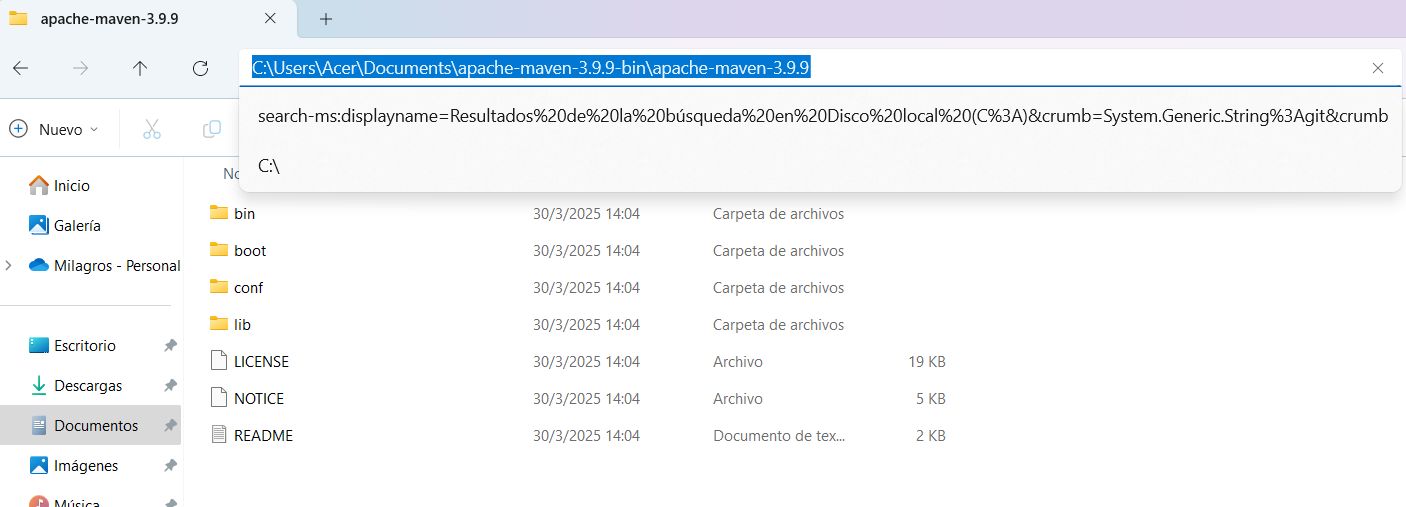
### *Instalación y Configuración de Maven*

El proyecto requería Maven para su ejecución. Se instalaron los archivos necesarios siguiendo estos pasos:  
[Apache Maven - Descarga](https://maven.apache.org/download.cgi)

Pasos:  
1️-Descargar el archivo **bin.zip**.  
2️-Extraer la carpeta en C:\Program Files\Apache Maven.  
3️- Seguir los pasos de para agregarlo al PATH:

Buscar la carpeta donde está Maven, en mi caso: C:\Users\Acer\Documents\apache-maven-3.9.9-bin\apache-maven-3.9.9

**Figura 14**

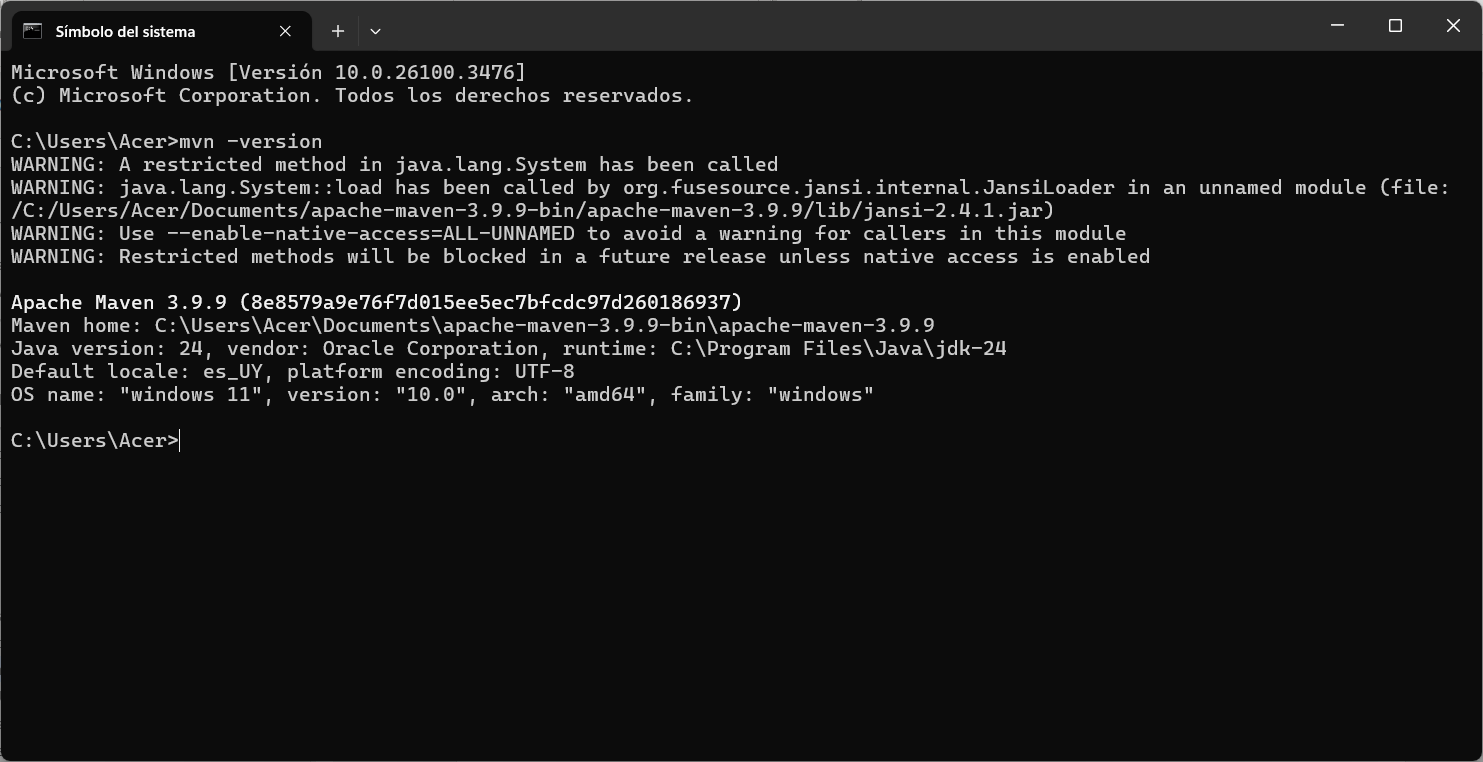
*Captura de pantalla del explorador de archivos mostrando el contenido de la carpeta apache-maven-3.9.9 ubicada en C:\Users\Acer\Documents\apache-maven-3.9.9-bin\.*

*Nota.* Elaboración propia

* Agregar la ruta de Maven a la variable de entorno **PATH.**
* Verificar la instalación desde la terminal con **mvn -version.**
* Pulsar Win + R, escribir **sysdm.cpl** y presionar Enter.
* Ir a **Opciones avanzadas** → **Variables de entorno**.
* En **Variables del sistema**, buscar Path y edítarlo.
* Agrega la ruta C:\Users\Acer\Documents\apache-maven-3.9.9-bin\apache-maven-3.9.9
* Guardar y reiniciar la terminal.

Verificar si quedo instalado desde cmd:

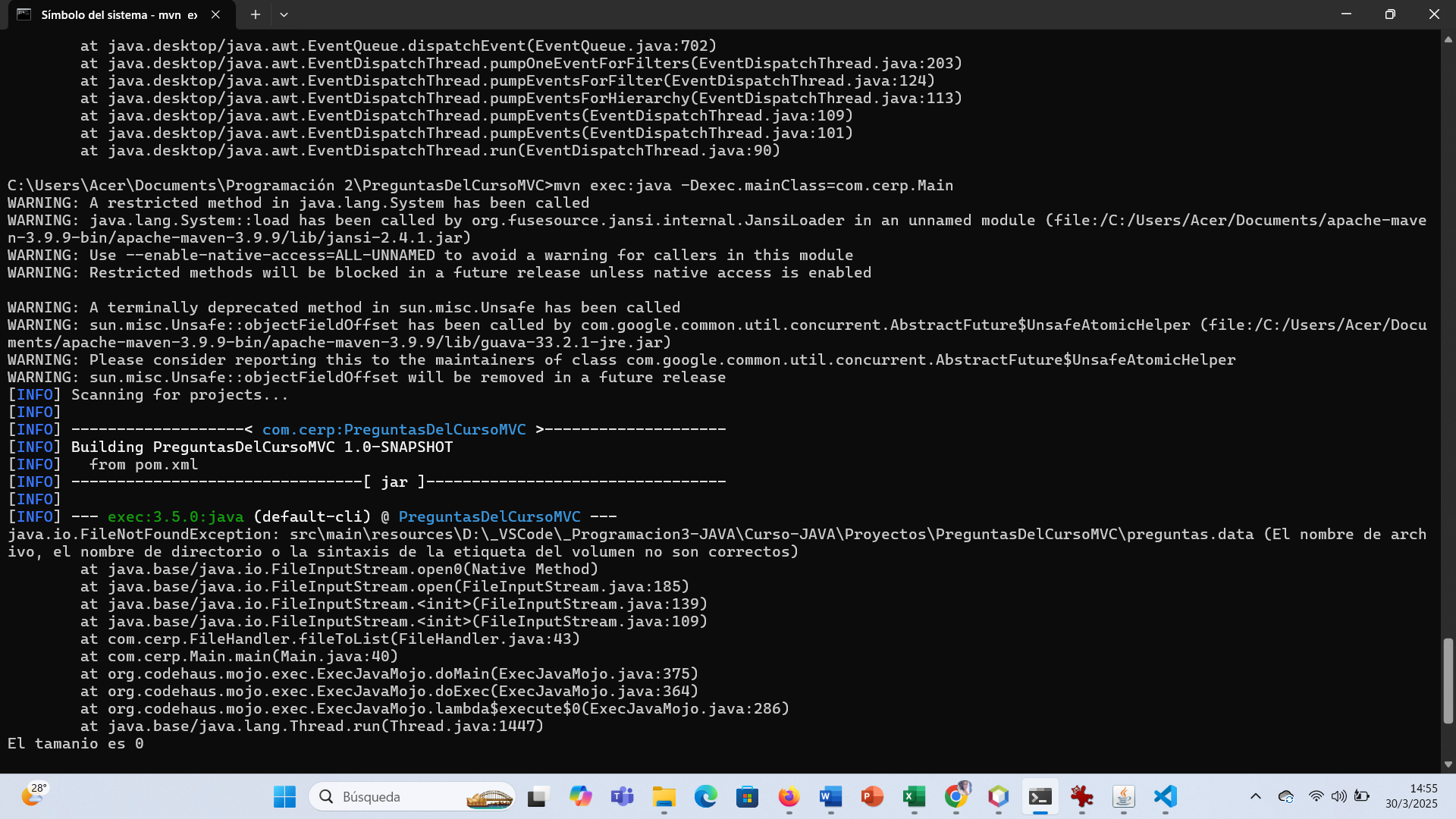
**Figura 15**

*Captura de pantalla de la línea de comandos mostrando la salida del comando mvn -version, confirmando la instalación de Apache Maven 3.9.9, la versión de Java (24) y el sistema operativo.*

*Nota.* Elaboración propia

El problema está en la ruta del archivo que intentas leer. La excepción que aparece: la ruta está mal construida. Parece que estás concatenando una ruta relativa (src/main/resources/) con una ruta absoluta (D:\\_VSCode\...), lo que genera un error.

**Figura 16**

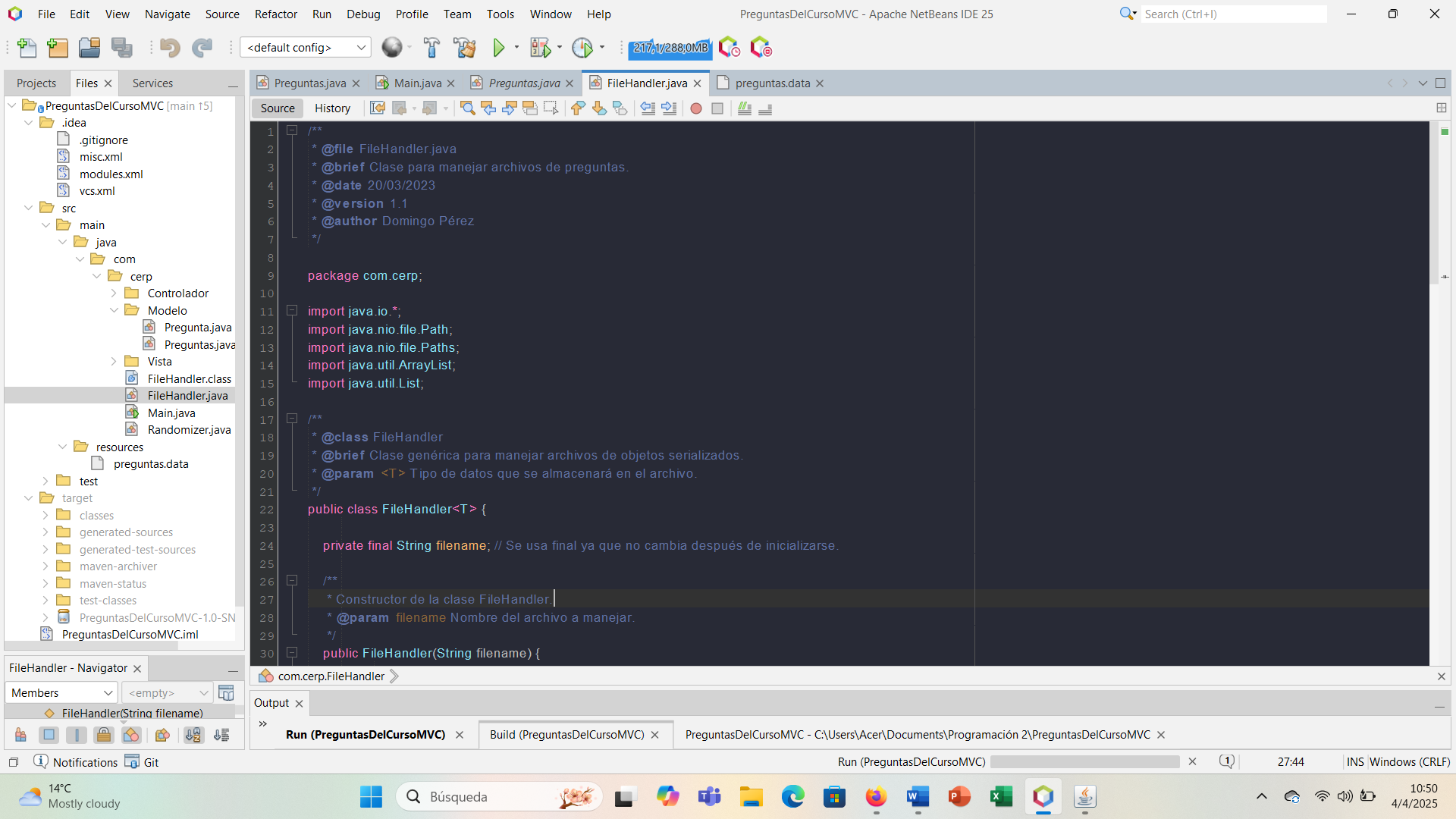
*Captura de pantalla de la línea de comandos mostrando la ejecución de la aplicación PreguntasDelCursoMVC con Maven, resultando en una excepción FileNotFoundException al intentar leer el archivo preguntas.data.*

*Nota.* Elaboración propia

### *Corrección de Advertencias en la Documentación Javadoc*

Se agregó la etiqueta **@param** en **FileHandler<T>** para documentar correctamente el tipo de datos genérico:

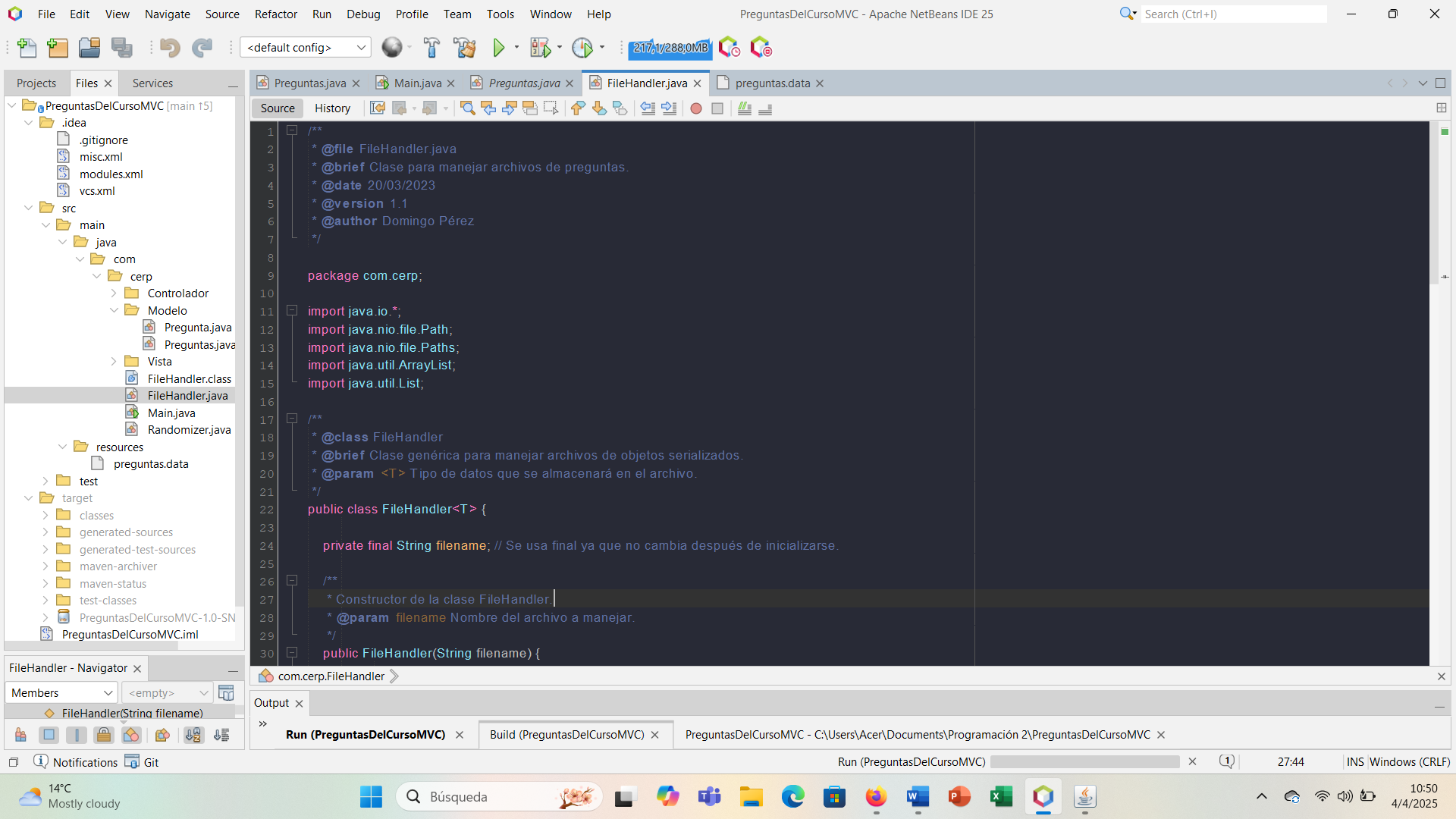
**Figura 17**

*Fragmento de código Java de la clase FileHandler dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración de la clase como genérica (FileHandler<T>), la importación de librerías relacionadas con la entrada/salida de archivos y la declaración del atributo filename.*

*Nota.* Elaboración propia

Además, se corrigió la advertenciasobre **filename,** declarándolo como **final:**

**Figura 18**

*Fragmento de código Java de la clase FileHandler dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración del atributo filename como privado y final de tipo String.*

*Nota.* Elaboración propia

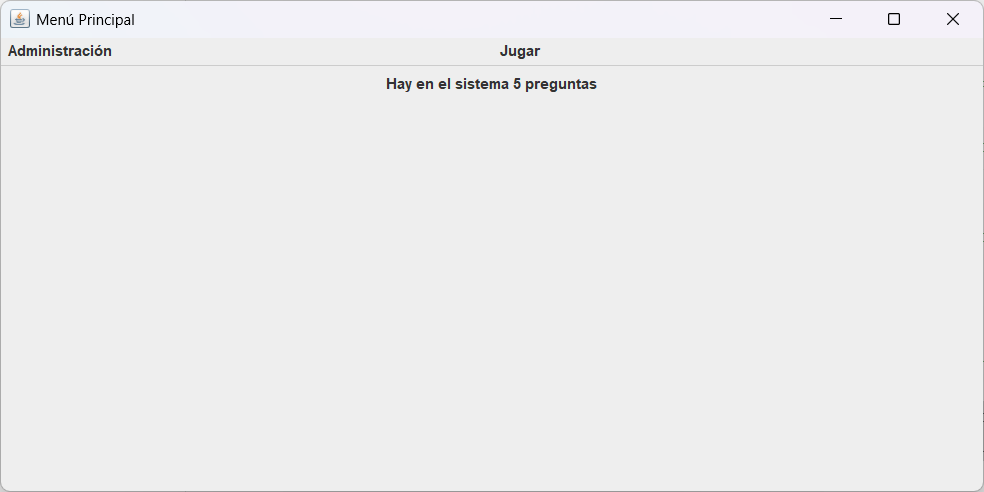
### *¿Qué Unidad Del Curso Seleccionaste Para Crear Las Preguntas?*

Se seleccionó la unidad "Introducción a la Programación Orientada a Objetos". Las preguntas fueron creadas en base a los siguientes documentos:

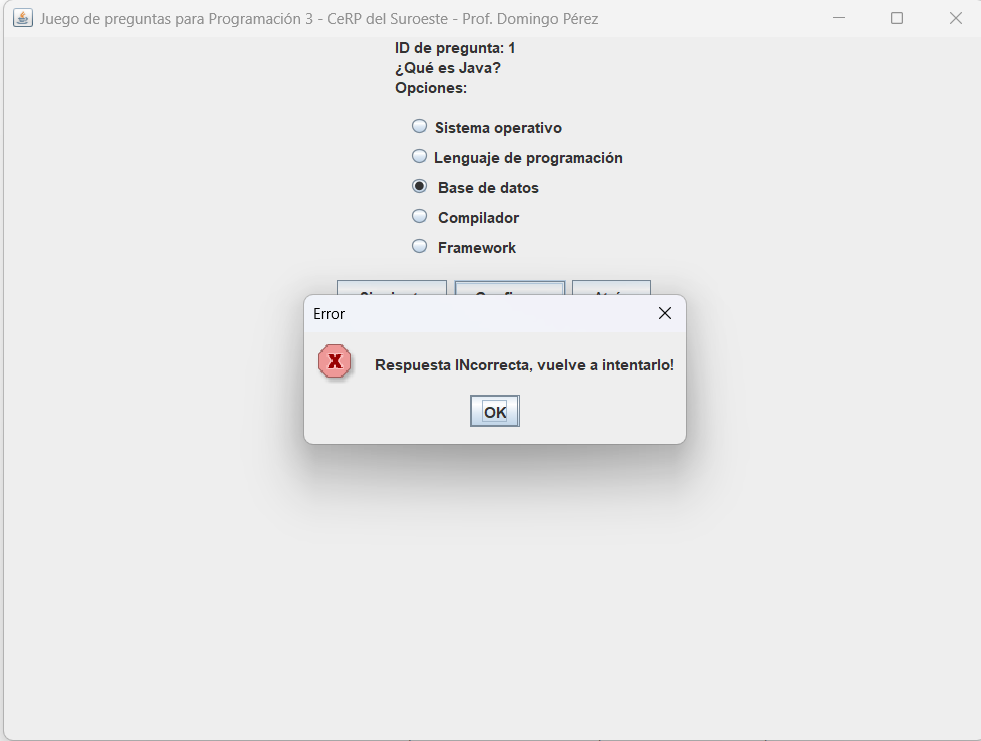
1. "¿Qué es Java?" - Esta información se encuentra principalmente en el PDF "1.2+Conceptos+POO.+Caracteristicas+Java.pdf", en el que Java se describe como un lenguaje de programación orientado a objetos utilizado para aplicaciones comerciales y en línea.
2. "¿Qué es la Programación Orientada a Objetos?" - Esta pregunta se relaciona con el PDF "1.1+Introduccion+a+POO.pdf", que describe la POO como "un paradigma de programación que busca simplificar y organizar el desarrollo de software al abordarlo desde una perspectiva basada en objetos y sus interacciones".
3. "¿Qué es una clase en Java?" - Este concepto aparece en ambos PDFs, pero se explica más directamente en "1.2+Conceptos+POO.+Caracteristicas+Java.pdf" donde se habla de clases como modelos o plantillas para crear objetos.
4. "¿Qué es un objeto en Java?" - Esta información se encuentra en ambos PDFs. En "1.1+Introduccion+a+POO.pdf" se define: "Un objeto es una entidad que encapsula tanto datos como comportamientos".[[2]](#footnote-2)
5. "¿Qué es la herencia en Java?"[[3]](#footnote-3) - Esta pregunta se relaciona directamente con el PDF "1.2+Conceptos+POO.+Caracteristicas+Java.pdf", que describe la herencia como "un mecanismo mediante el cual una clase puede heredar atributos y comportamientos de otra clase" y menciona que permite "una mayor organización y reutilización de código".

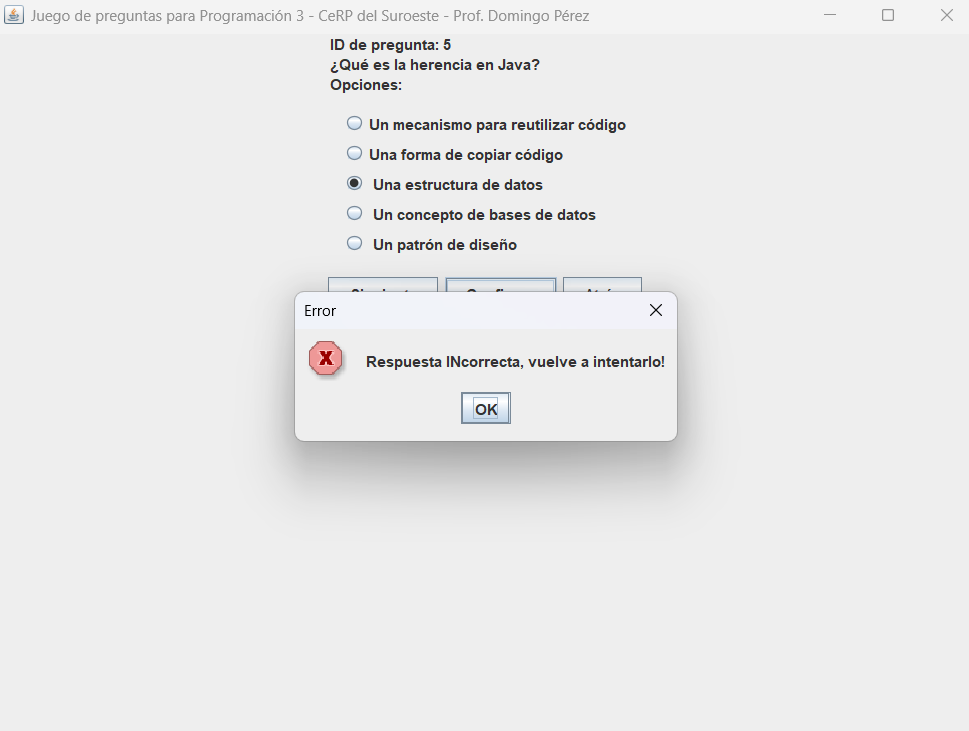
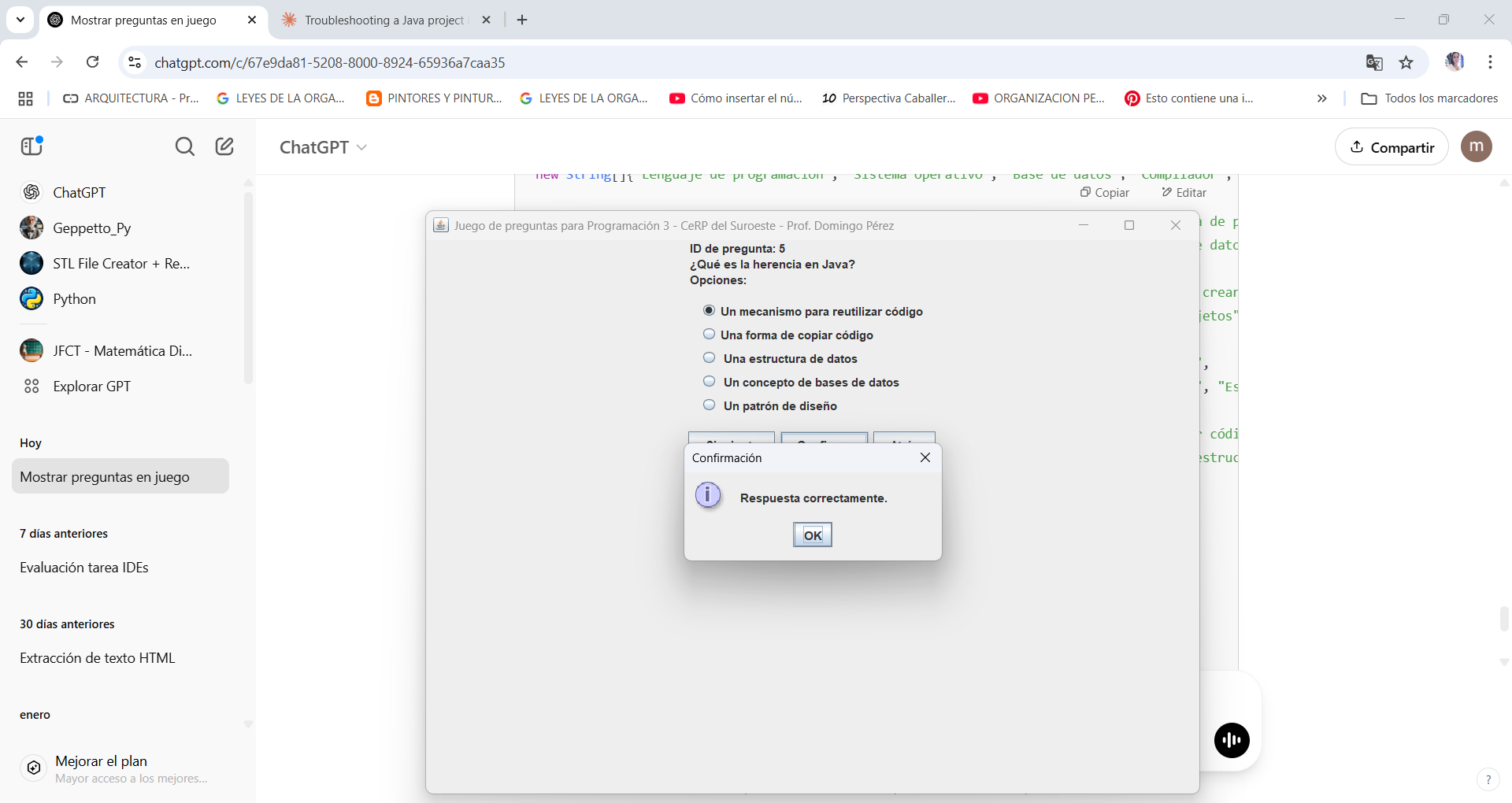
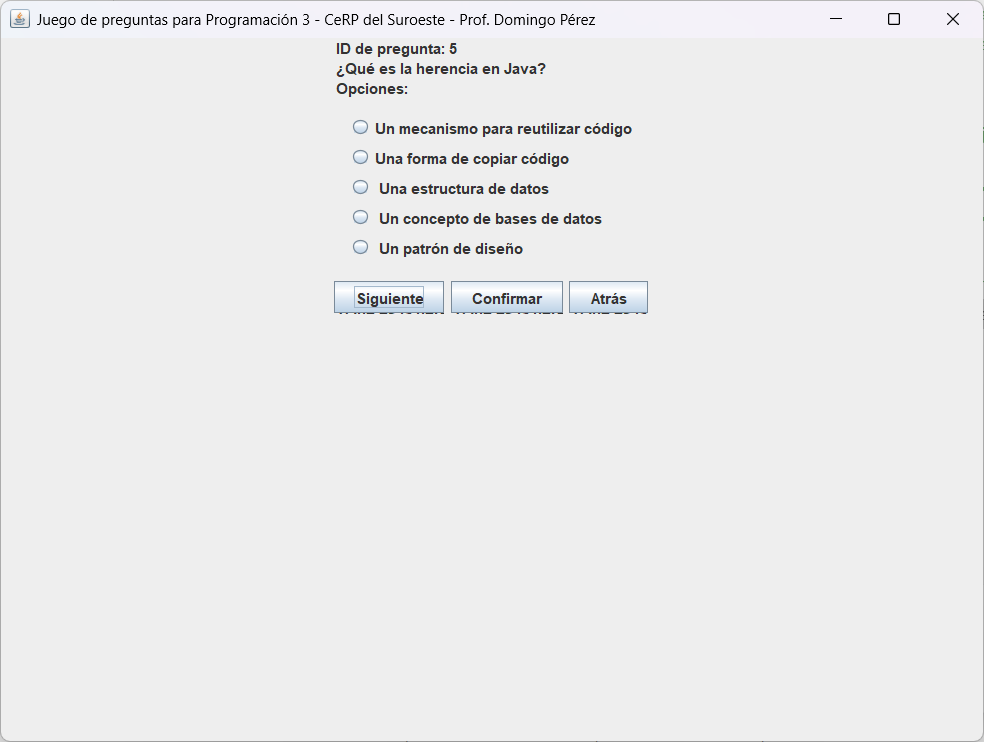
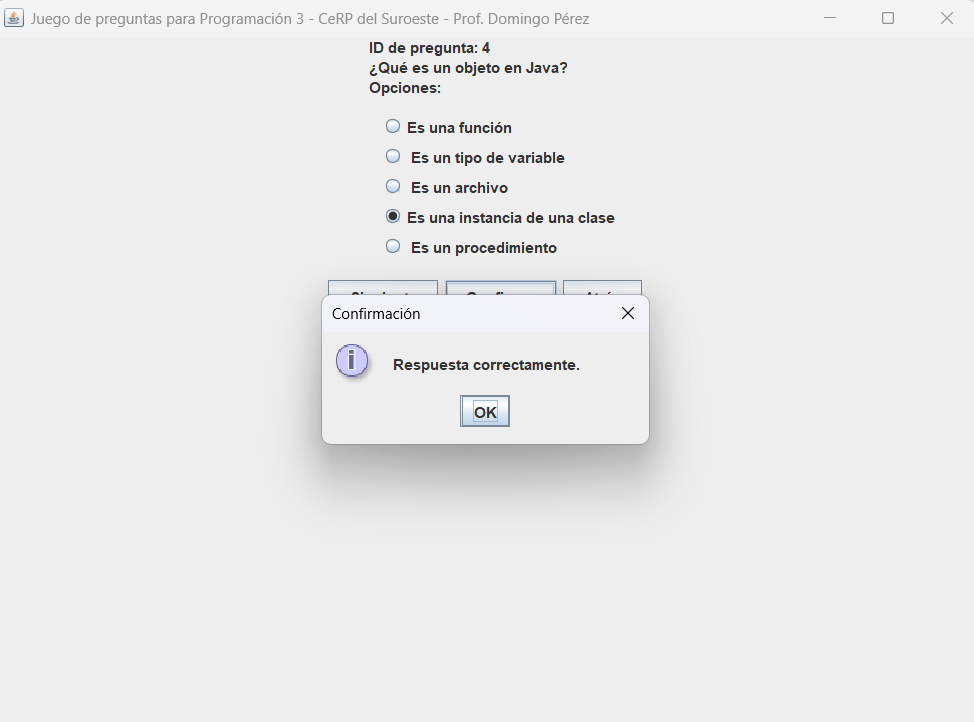
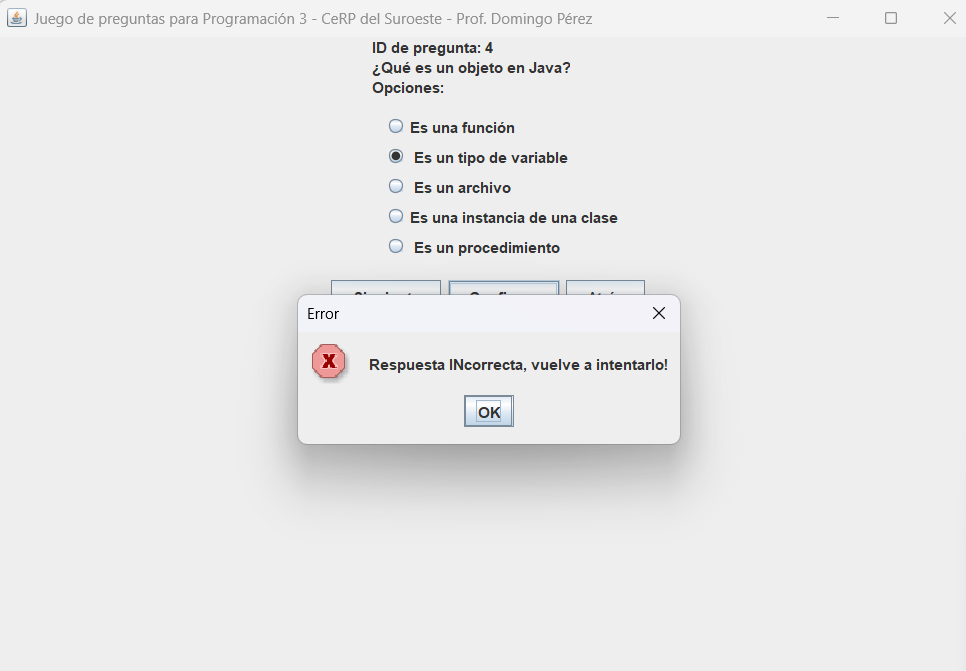
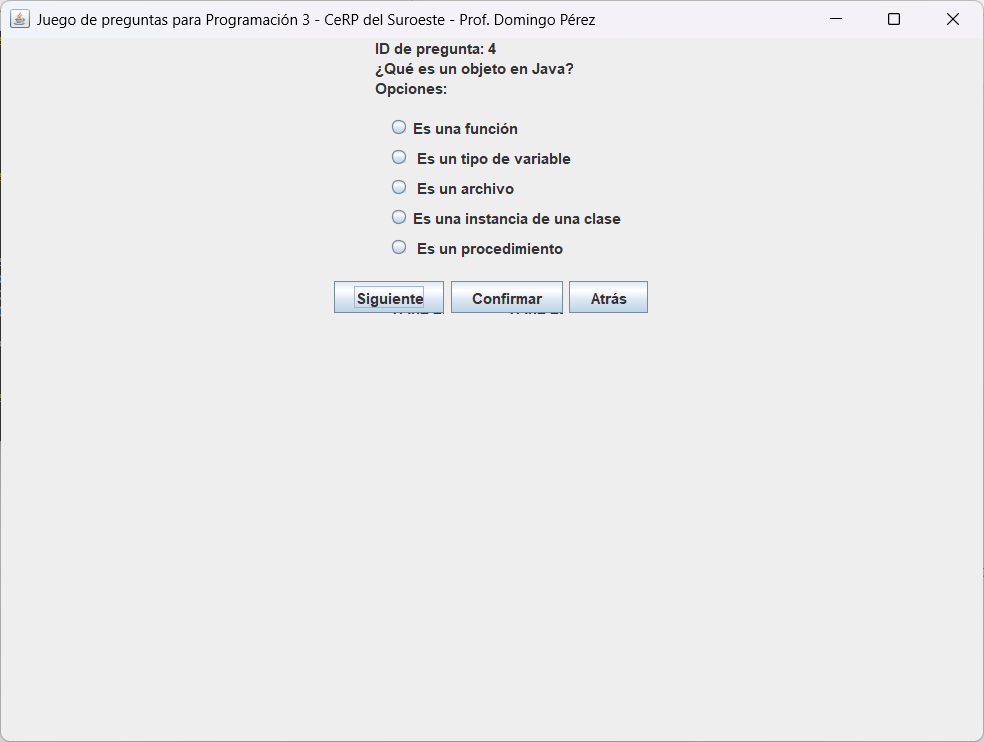
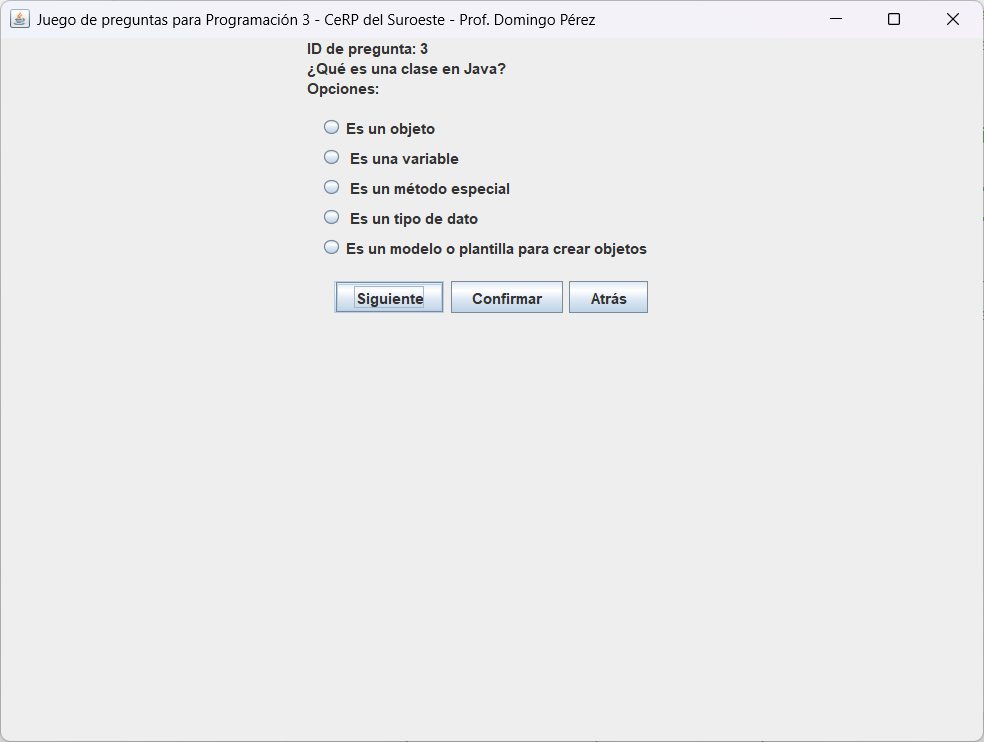
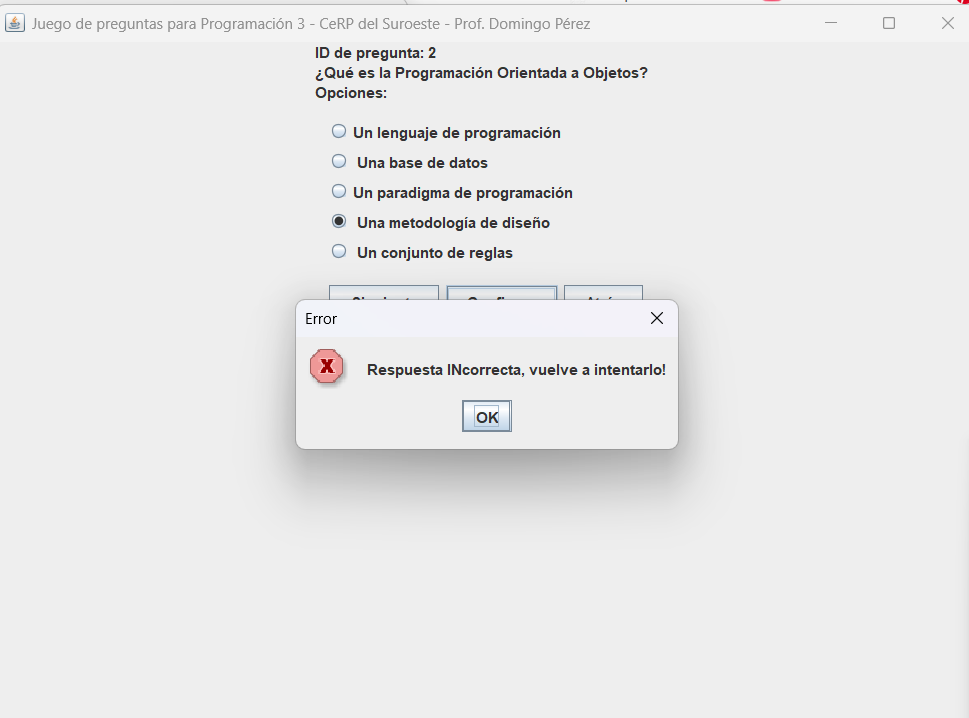
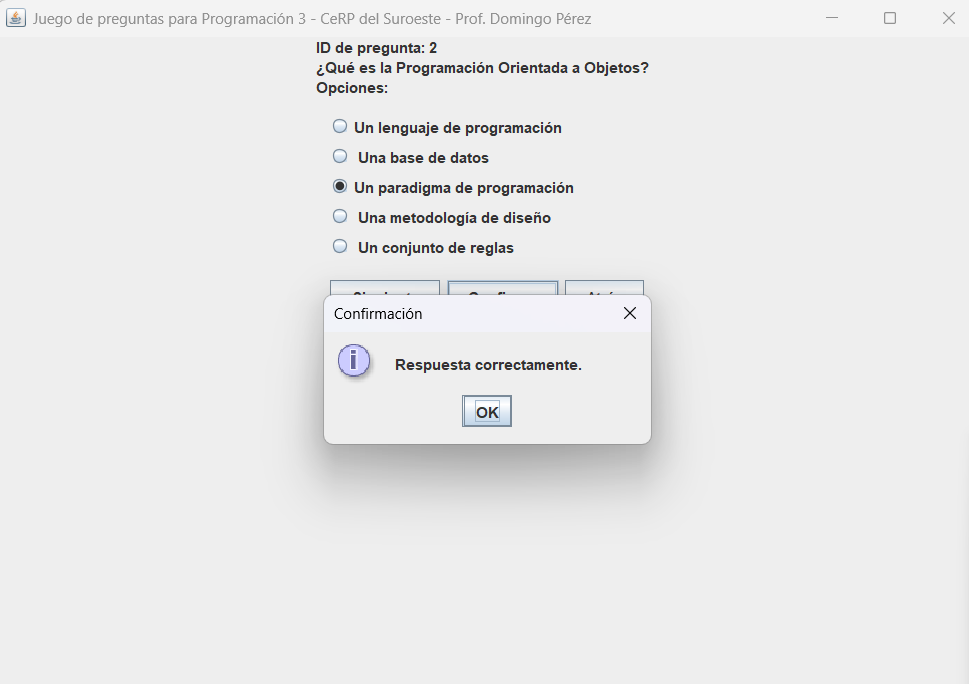
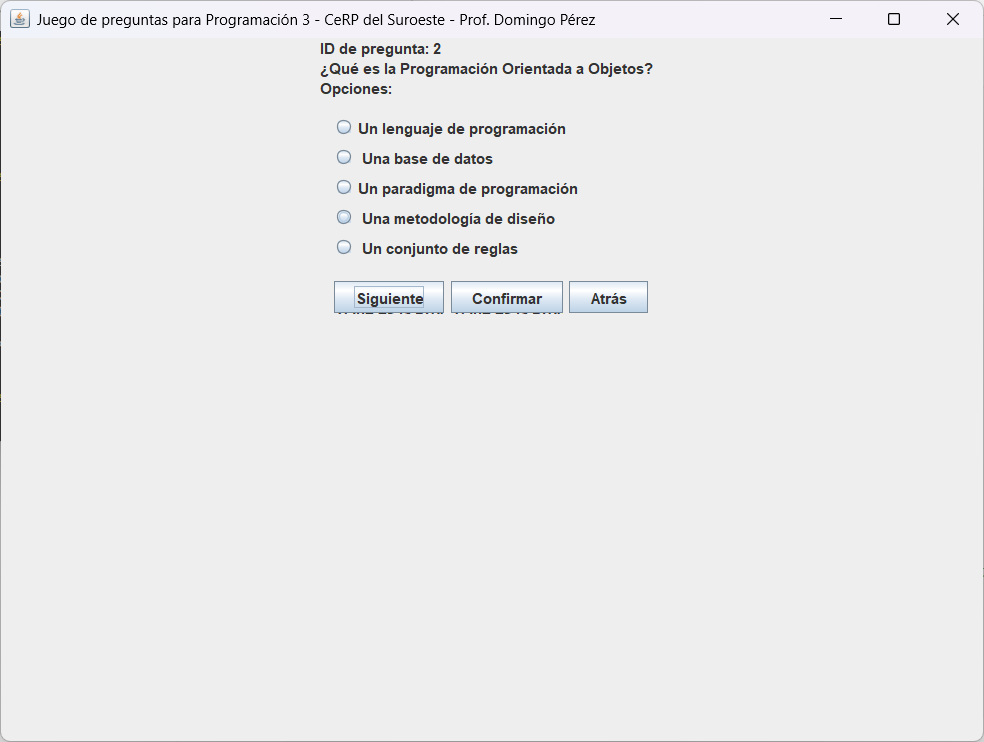
### *Capturas De Pantalla Propias Mostrando El Funcionamiento De Tu Versión Del Proyecto.*

**Figura 19**

*Captura de pantalla de la interfaz de la aplicación PreguntasDelCursoMVC, mostrando el menú principal con la opción "Jugar" seleccionada y el mensaje "Hay en el sistema 5 preguntas".*

*Nota.* Elaboración propia

A continuación, se muestra, captura de pantalla de la aplicación **PreguntasDelCursoMVC** durante el juego, mostrando una pregunta con opciones de respuesta seleccionadas y un mensaje de error indicando una respuesta incorrecta o correctas:



*Análisis Personal Sobre Las Dificultades Encontradas Y Cómo Las Resolviste.*

Durante el desarrollo del proyecto basado en la Programación Orientada a Objetos (POO), se presentaron diversas dificultades técnicas y conceptuales que requirieron un análisis detallado para su solución.

## **1. Modificación De La Clase Preguntas.java**

Inicialmente, la clase Preguntas.java no contenía preguntas predefinidas, lo que obligaba a cargarlas manualmente mediante métodos. Para corregir esto, se modificó el constructor de la clase, agregando preguntas de ejemplo con opciones y asegurando su correcta inicialización. Esto permitió que la lista de preguntas estuviera disponible desde el inicio de la ejecución del programa. Además, inicialmente las preguntas no aparecían de forma automática y tenían que cargarse manualmente. Se solucionó modificando el código para que tomara las preguntas automáticamente desde **Preguntas.java.**

*Dificultad:* El código ejecutable estaba fuera de un método o constructor, lo que generaba un error de sintaxis.

*Solución*: Se movieron las instrucciones de inicialización dentro del constructor **Preguntas()**, garantizando así que la estructura del código fuera válida y funcional.

#### **2. Implementación Correcta De La Interfaz List<Pregunta>**

Se intentó que la clase Preguntas implementara la interfaz List, pero esto generó múltiples errores, ya que no se sobrescribieron correctamente todos los métodos requeridos por la interfaz.

*Dificultad:* La implementación incompleta generaba excepciones **UnsupportedOperationException** en métodos como **size()** e **isEmpty()**.

*Solución:* Se eliminaron los métodos no implementados y se corrigieron los existentes para delegar correctamente en **listaPreguntas**, asegurando el correcto funcionamiento de las operaciones básicas de listas

## **3. Manejo De Rutas Y Acceso A Archivos**

El programa no encontraba el archivo **preguntas.data**, lo que impedía la correcta lectura y escritura de datos.

*Dificultad:* El archivo estaba ubicado en **src/main/resources/**, pero el código intentaba acceder a una ruta absoluta incorrecta.

*Solución:* Se utilizó un **ClassLoader** para acceder al archivo desde la carpeta de recursos y se modificó el constructor de FileHandler para manejar rutas relativas correctamente:

**InputStream inputStream =**

**getClass().getClassLoader().getResourceAsStream("preguntas.data");**

Esto permitió que el programa encontrara el archivo sin importar desde qué directorio se ejecutara.

## **4. Errores Con Maven Y Configuración Del Entorno**

El proyecto no compilaba debido a problemas con Maven, lo que impedía su ejecución.

*Dificultad:* Faltaba la instalación y configuración correcta de Maven en el sistema.

*Solución:* Se descargó e instaló Maven siguiendo estos pasos:

1. Descargar el archivo bin.zip desde el sitio oficial de Apache Maven.
2. Extraer la carpeta en C:\Program Files\Apache Maven.
3. Agregar la ruta de Maven a la variable de entorno PATH.
4. Verificar la instalación con mvn -version.

## **5. Documentación Y Buenas Prácticas**

La modificación del código incluyó mejoras en la estructura del proyecto, la claridad de los nombres de variables y métodos, y la incorporación de comentarios que explican el propósito de cada bloque de código. Estas buenas prácticas están alineadas con los principios de la ingeniería del software. Según Pressman (2008), el desarrollo de software es un proceso completo que requiere la aplicación de métodos, técnicas y herramientas de ingeniería para planificar, diseñar, implementar, probar y mantener sistemas que satisfagan las necesidades del usuario (pp. 12, 13).

Desde una perspectiva educativa, enseñar POO implica más que transmitir conceptos técnicos: requiere fomentar el pensamiento lógico, la abstracción y la resolución de problemas. La POO puede resultar abstracta para algunos estudiantes, especialmente si no se acompaña de ejemplos concretos y visuales. Es fundamental, entonces, diseñar experiencias de aprendizaje activas, como simulaciones o proyectos prácticos, que permitan interiorizar estos conceptos de forma significativa.

Además, este paradigma se alinea con enfoques pedagógicos constructivistas, ya que invita al estudiante a construir soluciones propias basadas en una lógica interna coherente. Introducir a los estudiantes a la POO desde una edad temprana puede fortalecer sus competencias digitales, analíticas y creativas.

### ****Conclusión****

El desarrollo del proyecto implicó enfrentar múltiples dificultades relacionadas con la estructura del código, la implementación de interfaces, la gestión de archivos y la configuración del entorno. Cada problema fue abordado con una estrategia específica, aplicando principios de la POO y prácticas de programación. Estas soluciones no solo permitieron corregir errores, sino que también mejoraron la organización y eficiencia del código.

Explorar los fundamentos de la POO a través de un caso práctico permitió evidenciar la importancia de un diseño orientado a objetos para lograr soluciones modulares, flexibles y sostenibles en el tiempo. Además, se comprendió que más allá de aprender un lenguaje como Java, es fundamental desarrollar una mentalidad orientada a objetos. Como afirman Pino, Martínez y Vergara (2021), la Programación Orientada a Objetos se fundamenta en la definición de clases y objetos, permitiendo la abstracción y encapsulación de datos, la herencia para la reutilización del código y el polimorfismo para adaptar comportamientos de manera flexible.

Comprender los fundamentos de la Programación Orientada a Objetos no solo es crucial para el desarrollo profesional en el campo del software, sino que también ofrece herramientas cognitivas poderosas para analizar y modelar el mundo. La enseñanza efectiva de este paradigma debe combinar el rigor técnico con estrategias didácticas que conecten con los intereses y experiencias del estudiantado.

# **Referencias bibliográficas**

Abrirllave. (s.f.). *Tipos de datos primitivos en Java*. Recuperado el 26 de marzo de 2025. Disponible desde internet: <https://www.abrirllave.com/java/tipos-de-datos-primitivos.php>

Bermúdez, J.B. (2012) *Programación orientada a Objetos con Java* - ETSISI-UPM, Escuela Técnica Superior de Ingeniería de SISTEMAS INFORMÁTICOS Universidad Politécnica de Madrid. Edited by L. Fernández. Available at: <https://www.etsisi.upm.es/sites/default/files/curso_2013_14/MASTER/MIW.JEE.POOJ.pdf>

Booch, G. (2007). \*Object-Oriented Analysis and Design with Applications\* (3rd ed.). Addison-Wesley.

Ceballos, Fco. Javier, (2011). *Java 2: Curso de Programación, 4a. ed*., Alfaomega, México

*Concepto de la programación en java.* (s.f). Recuperado el 26 de marzo de 2025. Disponible desde internet: <http://todojava.awardspace.com/>

Eckel, B. (2009). *Piensa en Java*. Cuarta Edición. Prentice-Hall. Recuperado el 26 de marzo de 2025. Disponible desde internet: https://www.mfbarcell.es/docencia\_uned/poo/material\_complementario/piensaenjava/Piensa%20en%20Java%20-%20Bruce%20Eckel.pdf

Gómez Jiménez, E. (2012). *Desarrollo de Software con NetBeans 7.1: ¡Programe para escritorio, Web y dispositivos móviles!* México*:* Alfaomega. https://elhacker.info/manuales/Lenguajes%20de%20Programacion/Java/-Desarrollo-de-Software-Con-NetBeans-7-1.pdf

López, José (2011). *Domine JavaScript - 3ª ed.,* México, Alfaomega

Morales, R. (2014). *Lenguajes de programación: ¿qué son y para qué sirven?.* Recuperado el 26 de marzo de 2025. Disponible desde internet en: <https://colombiadigital.net/actualidad/articulos-informativos/item/7669-lenguajes-de-programacion-que-son-y-para-que-sirven.html>

*NetBeans.* Enlace de descarga: <https://netbeans.apache.org/download/index.html>

Oracle. (2023). What is Object-Oriented Programming? <https://www.oracle.com/java/technologies/oop.html>

Pérez, J., Merino, M. (2009). *Definición de: Lenguaje de programación*. Recuperado el 28 de marzo de 2025. Disponible desde internet: <https://definicion.de/lenguaje-de-programacion/>

Pino, J. H., Martínez, P. M., & Vergara, J. A. (2021). Fundamentos de la programación en Java. Estructuras de Control e Introducción a la Programación Orientada a Objetos [PDF]. Editorial Círculo Rojo. <https://www.uv.mx/personal/pmartinez/files/2021/03/fundamentosdelaprogramacionenjava_completo2021.pdf>

Pressman, R. (2010). Ingeniería del software Un enfoque práctico séptima edición, Ph.D. University of Connecticut. McGraw-Hil

TIOBE. (2019). *TIOBE Index for April 2019*. Recuperado el 26 de marzo de 2025. Disponible desde internet:  <https://www.tiobe.com/tiobe-index/>

Umbaugh, J., Jacobson, I., & Booch, G. (2005). *El Lenguaje Unificado de Modelado Manual de Referencia Segunda Edición*. Madrid: Pearson

Universidad Don Bosco. (2019). *Guía 1: Introducción a la programación avanzada en Java*. Universidad Don Bosco. <https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/java-avanzado/2019/i/guia-1.pdf>

1. En la POO, una **clase** es una plantilla o molde que define las propiedades (atributos) y comportamientos (métodos) de un objeto. Un **objeto** es una instancia concreta de una clase. [↑](#footnote-ref-1)
2. El encapsulamiento protege el estado interno de un objeto al restringir el acceso directo a sus atributos. Se logra mediante modificadores de acceso (como private en Java o \_ en Python). Esto promueve la modularidad y evita errores por manipulaciones indebidas. [↑](#footnote-ref-2)
3. Según Pressman (2019), la herencia permite reutilizar código de forma jerárquica.La **herencia** permite a una clase derivada reutilizar atributos y métodos de una clase base. Es útil para crear jerarquías lógicas [↑](#footnote-ref-3)