



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
Tarea: Explorando Los Conceptos Fundamentales De La POO	4
1- Enlace A Repositorio Personal Con El Proyecto Modificado:	4
2- Documenta detalladamente:	4
□ <i>¿Qué Modificaciones Realizaste En El Proyecto?</i>	4
<i>Corrección de Código Ejecutable Fuera de un Método o Constructor</i>	8
<i>Ejemplo de Clases y Objetos: Ejes estructurales de la POO</i>	9
<i>Corrección de la Implementación de Métodos con @Override</i>	10
<i>Corrección de la Implementación de List</i>	12
<i>Manejo de Errores con el Archivo "preguntas.data"</i>	12
<i>Ajustes en la Estructura del Proyecto</i>	13
<i>Instalación y Configuración de Maven</i>	16
<i>Corrección de Advertencias en la Documentación Javadoc</i>	19
□ <i>¿Qué Unidad Del Curso Seleccionaste Para Crear Las Preguntas?</i>	19
□ <i>Capturas De Pantalla Propias Mostrando El Funcionamiento De Tu Versión Del Proyecto.</i>	20
1. Modificación De La Clase Preguntas.java	25
3. Manejo De Rutas Y Acceso A Archivos	26
4. Errores Con Maven Y Configuración Del Entorno	27
5. Documentación Y Buenas Prácticas	27
Conclusión	28
Referencias bibliográficas	29



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

1- Enlace A Repositorio Personal Con El Proyecto Modificado:

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

2- 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

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, como se observa en la imagen a continuación:

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.

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.

```

Domingo1987 SUBIR MEJORAS 2025 X
Code Blame 186 lines (152 loc) · 5.47 KB
1 package com.cerp.modelo;
2
3 import java.io.Serializable;
4 import java.util.ArrayList;
5 import java.util.Collection;
6 import java.util.Iterator;
7 import java.util.List;
8 import java.util.ListIterator;
9
10 public class Preguntas implements Serializable, List<Pregunta> {
11     private List<Pregunta> listaPreguntas;
12
13     public Preguntas() {
14         this.listaPreguntas = new ArrayList<Pregunta>();
15     }
16
17     public List<Pregunta> getListaPreguntas() {
18         return listaPreguntas;
19     }
20
21     public void setListaPreguntas(List<Pregunta> listaPreguntas) {
22         this.listaPreguntas = listaPreguntas;
23     }
24
25     public void addPregunta(Pregunta pregunta) {

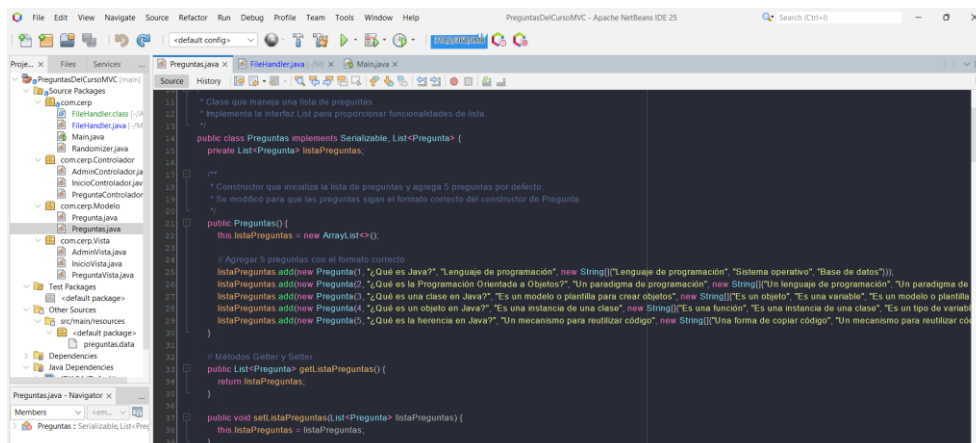
```

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. Como se observa en la Figura 2:

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.



```

1 // Clase que maneja una lista de preguntas.
2 // Implementa la interfaz List para proporcionar funcionalidades de lista.
3
4 public class Preguntas implements Serializable, List<Pregunta> {
5     private List<Pregunta> listaPreguntas;
6
7     /**
8      * Constructor que inicializa la lista de preguntas y agrega 5 preguntas por defecto.
9      * Se modificó para que las preguntas sigan el formato correcto del constructor de Pregunta.
10      */
11     public Preguntas() {
12         this.listaPreguntas = new ArrayList<>();
13
14         // Agregar 5 preguntas con el formato correcto
15         listaPreguntas.add(new Pregunta(1, "¿Qué es Java?", "Lenguaje de programación", new String[]{"Lenguaje de programación", "Sistema operativo", "Base de datos"}));
16         listaPreguntas.add(new Pregunta(2, "¿Qué es la Programación Orientada a Objetos?", "Un paradigma de programación", new String[]{"Un lenguaje de programación", "Un paradigma de programación", "Un modelo o plantilla para crear objetos", "Es un objeto", "Es una variable", "Es un modelo o plantilla"}));
17         listaPreguntas.add(new Pregunta(3, "¿Qué es una clase en Java?", "Es una instancia de una clase", new String[]{"Es una función", "Es una instancia de una clase", "Es un tipo de variable", "Es un tipo de variable"}));
18         listaPreguntas.add(new Pregunta(4, "¿Qué es la herencia en Java?", "Un mecanismo para reutilizar código", new String[]{"Una forma de copiar código", "Un mecanismo para reutilizar código", "Un mecanismo para reutilizar código", "Un mecanismo para reutilizar código"}));
19
20         // Métodos Getters y Setters
21     }
22
23     public List<Pregunta> getListaPreguntas() {
24         return listaPreguntas;
25     }
26
27     public void setListaPreguntas(List<Pregunta> listaPreguntas) {
28         this.listaPreguntas = listaPreguntas;
29     }
30
31     public void addPregunta(Pregunta pregunta) {

```

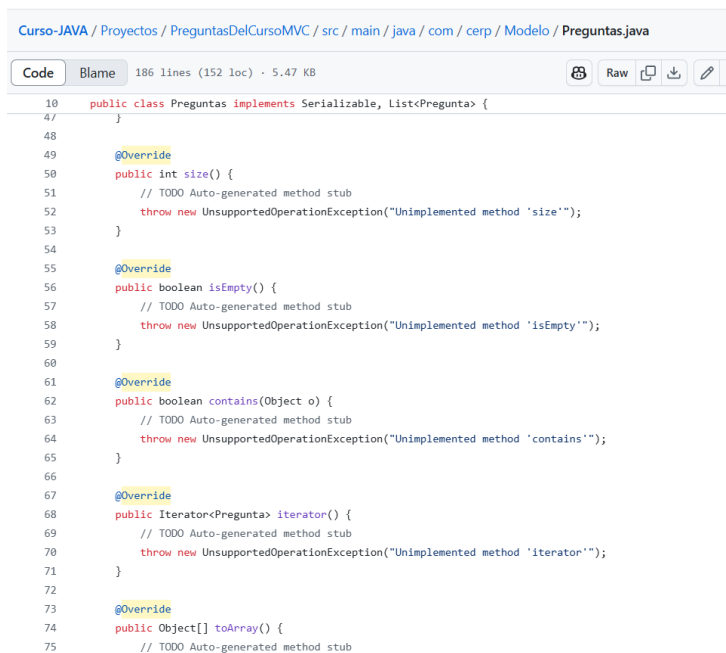
Nota. Elaboración propia.

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 (ver Figura 3)

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).



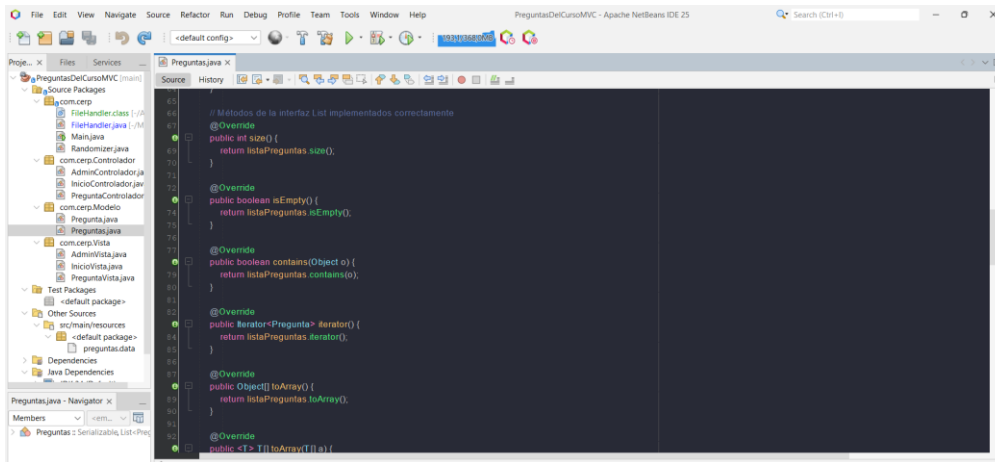
```
Curso-JAVA / Proyectos / PreguntasDelCursoMVC / src / main / java / com / cerp / Modelo / Preguntas.java
Code Blame 186 lines (152 loc) · 5.47 KB
10 public class Preguntas implements Serializable, List<Pregunta> {
47 }
48
49 @Override
50 public int size() {
51     // TODO Auto-generated method stub
52     throw new UnsupportedOperationException("Unimplemented method 'size'");
53 }
54
55 @Override
56 public boolean isEmpty() {
57     // TODO Auto-generated method stub
58     throw new UnsupportedOperationException("Unimplemented method 'isEmpty'");
59 }
60
61 @Override
62 public boolean contains(Object o) {
63     // TODO Auto-generated method stub
64     throw new UnsupportedOperationException("Unimplemented method 'contains'");
65 }
66
67 @Override
68 public Iterator<Pregunta> iterator() {
69     // TODO Auto-generated method stub
70     throw new UnsupportedOperationException("Unimplemented method 'iterator'");
71 }
72
73 @Override
74 public Object[] toArray() {
75     // TODO Auto-generated method stub
```

Nota. Elaboración propia.

Ahora, estos métodos fueron implementados correctamente, delegando su funcionalidad a **listaPreguntas**, como muestra la siguiente imagen:

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.



```

// Métodos de la interfaz List implementados correctamente
@Override
public int size() {
    return listaPreguntas.size();
}

@Override
public boolean isEmpty() {
    return listaPreguntas.isEmpty();
}

@Override
public boolean contains(Object o) {
    return listaPreguntas.contains(o);
}

@Override
public Iterator<Pregunta> iterator() {
    return listaPreguntas.iterator();
}

@Override
public Object[] toArray() {
    return listaPreguntas.toArray();
}

@Override
public <T> T[] toArray(T[] a) {
    return listaPreguntas.toArray(a);
}

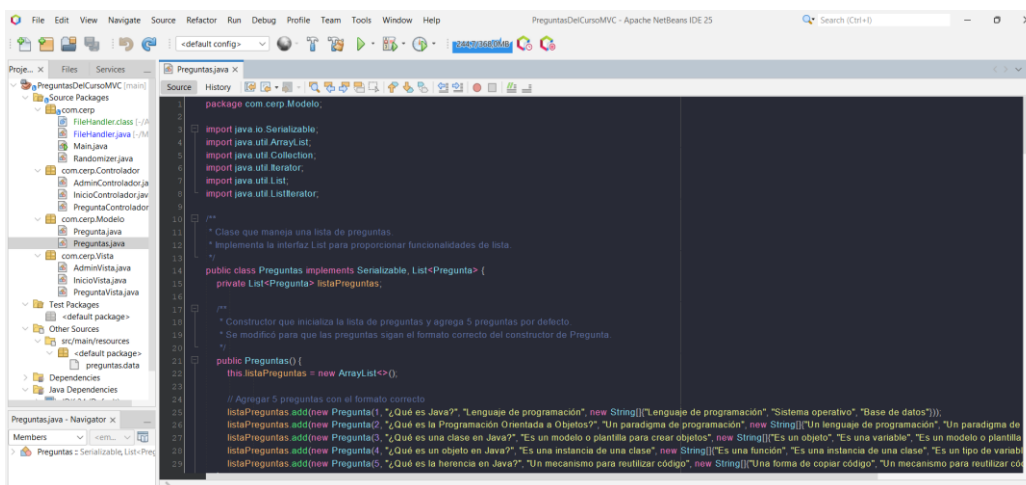
```

Nota. Elaboración propia.

Se añadieron comentarios explicativos para mejorar la comprensión del código como se observa en la Figura 5:

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.



```

package com.cerp.modelo;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

/**
 * Clase que maneja una lista de preguntas.
 * Implementa la interfaz List para proporcionar funcionalidades de lista.
 */
public class Preguntas implements Serializable, List<Pregunta> {
    private List<Pregunta> listaPreguntas;

    /**
     * Constructor que inicializa la lista de preguntas y agrega 5 preguntas por defecto.
     * Se modificó para que las preguntas sigan el formato correcto del constructor de Pregunta.
     */
    public Preguntas() {
        this.listaPreguntas = new ArrayList<>();

        // Agregar 5 preguntas con el formato correcto
        listaPreguntas.add(new Pregunta(1, "¿Qué es Java?", "Lenguaje de programación", "Sistema operativo", "Base de datos"));
        listaPreguntas.add(new Pregunta(2, "¿Qué es la Programación Orientada a Objetos?", "Un paradigma de programación", "new String()", "Un lenguaje de programación", "Un paradigma de listaPreguntas.add(new Pregunta(3, \"¿Qué es una clase en Java?\", \"Es un modelo o plantilla para crear objetos\", \"new String()\"Es un objeto\", \"Es una variable\", \"Es un modelo o plantilla listaPreguntas.add(new Pregunta(4, \"¿Qué es un objeto en Java?\", \"Es una instancia de una clase\", \"new String()\"Es una función\", \"Es una instancia de una clase\", \"Es un tipo de variable listaPreguntas.add(new Pregunta(5, \"¿Qué es la herencia en Java?\", \"Un mecanismo para reutilizar código\", \"new String()\"Una forma de copiar código\", \"Un mecanismo para reutilizar código\"));
    }
}

```



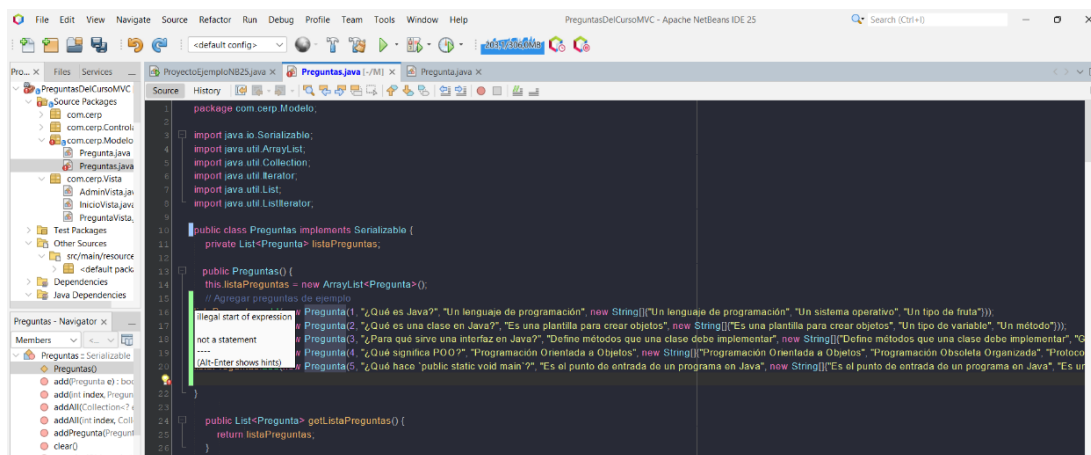
Nota. Elaboración propia.

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()**, como se muestra en la siguiente imagen:

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.

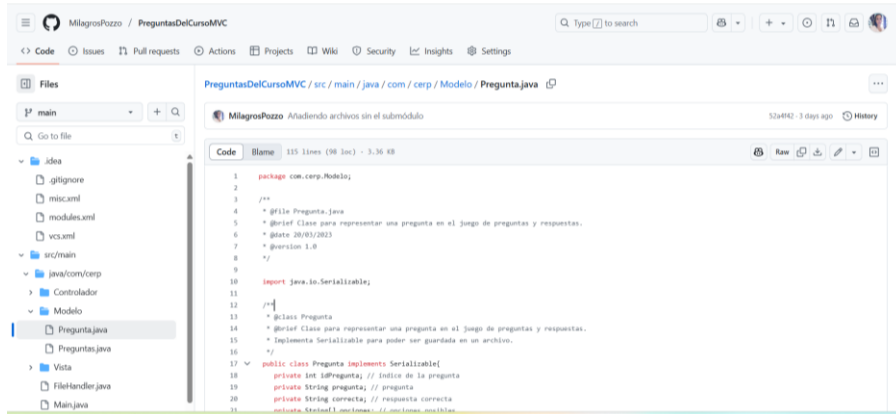


Nota. Elaboración propia.

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** como muestra la siguiente imagen:

Figura 7

Captura de pantalla del archivo Pregunta.java ubicado en la ruta src/main/java/com/serp/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



Encapsulamiento

Este principio consiste en restringir el acceso directo a ciertos componentes de un objeto, protegiendo así el estado interno del mismo. Mediante modificadores de acceso (**private**, **public**, **protected**), se promueve el uso de métodos para interactuar con los datos.

Esto fomenta la seguridad y la modularidad del software, permitiendo que el código sea menos propenso a errores y más fácil de mantener.

Herencia

La herencia permite que una clase derive de otra, heredando sus atributos y métodos. Esto favorece la reutilización del código y la organización jerárquica de las clases. Por ejemplo, una clase **Animal** puede servir de base para **Perro** y **Gato**, evitando repetir código común.

Es importante considerar el principio de sustitución de Liskov, que sugiere que las clases derivadas deben poder sustituir a sus clases base sin alterar el comportamiento del programa (Liskov, 1987).

Polimorfismo

El polimorfismo hace posible que múltiples clases respondan de manera diferente a una misma operación. Esto se logra mediante sobrecarga de métodos y sobrescritura. Por ejemplo, un método **hacerSonido()** puede tener implementaciones distintas en clases **Perro** y **Gato**, pero ser invocado de manera uniforme.

El polimorfismo promueve el diseño flexible y extensible, reduciendo la dependencia del código en clases específicas.

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`.

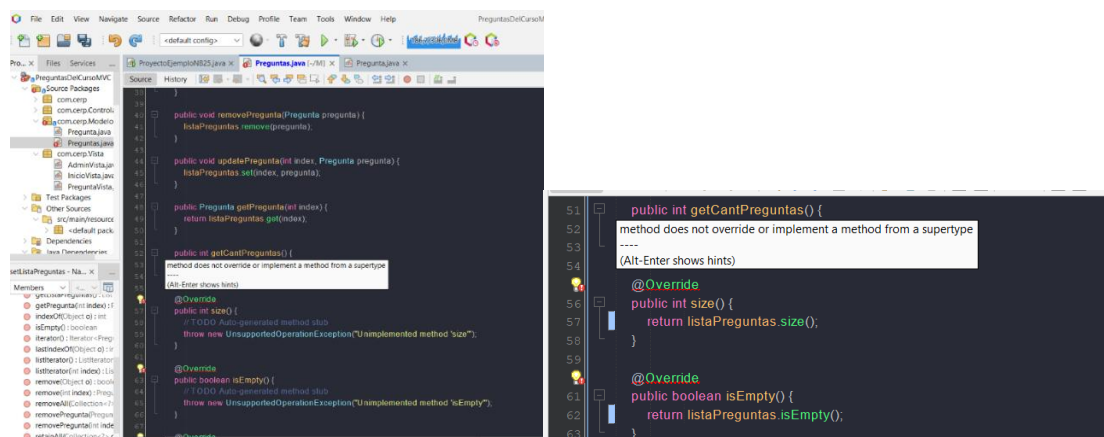
Dificultad: El código no tenía comentarios explicativos, lo que dificultaba su comprensión. Además, la anotación `@Override` se utilizaba en métodos que no sobrescribían ninguna superclase.

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, como se muestra en la siguiente Figura.

Figura 8

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



Nota. Elaboración propia.

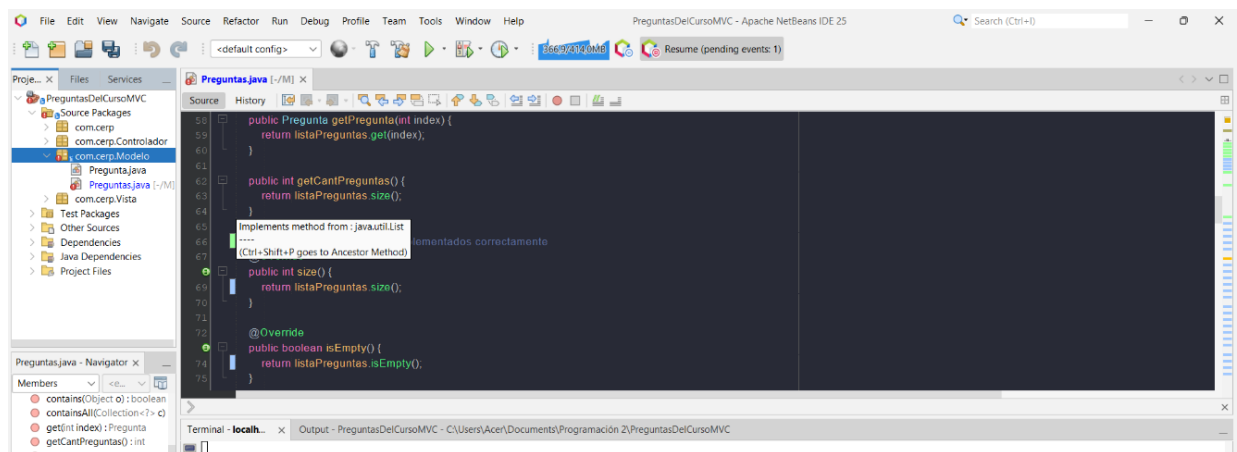
Para corregirlo:

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 (ver Figura 9)

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:

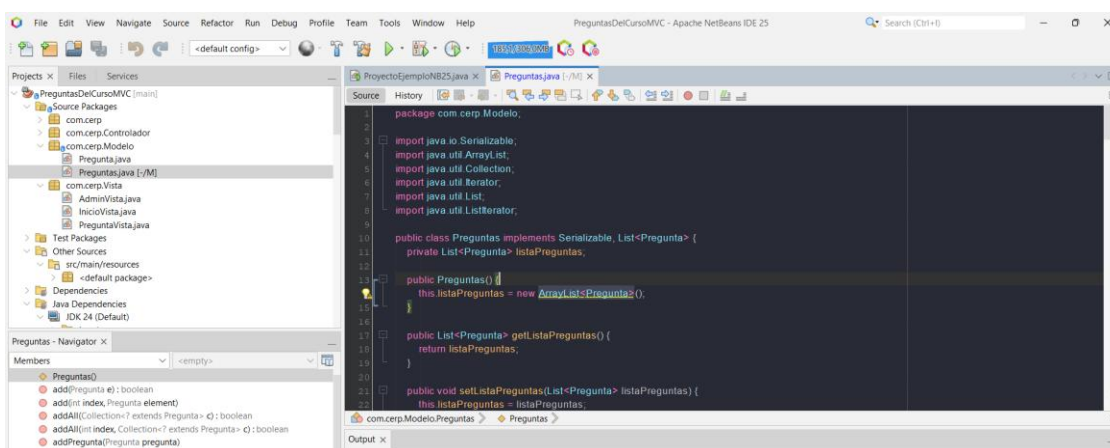
1. Se actualizó la ruta del archivo en el método **FileHandler.fileToList()** para usar una ruta relativa.
2. Se utilizó un cargador de clases para acceder al archivo dentro del directorio de recursos:

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

3. Se modificó **Main.java** para utilizar preguntas por defecto si **preguntas.data** estaba vacío.
4. 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** (Ver Figura 10)

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**.

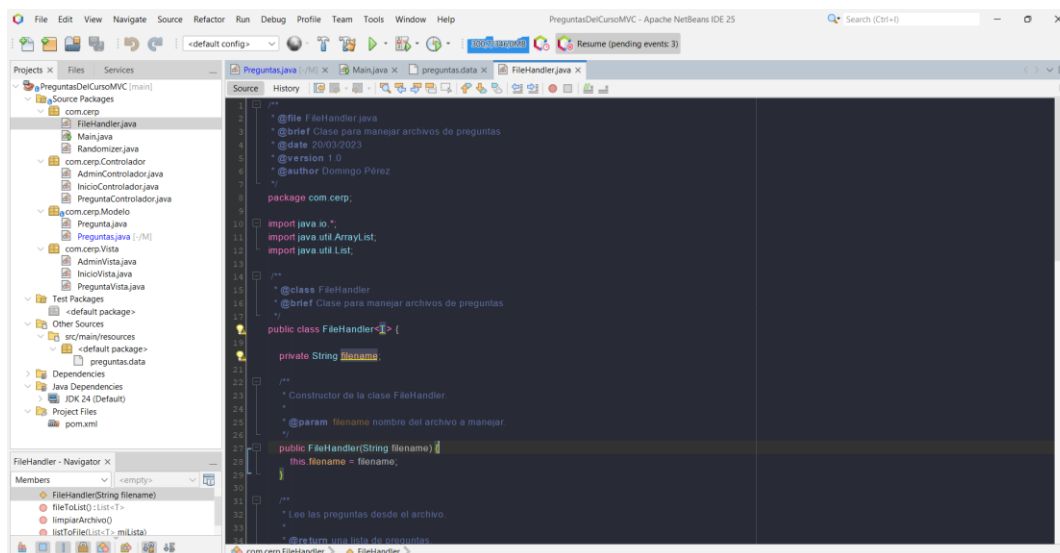
Allí se encuentra **FileHandler.java**.

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** como se muestra en la siguientes imágenes:

Figura 11

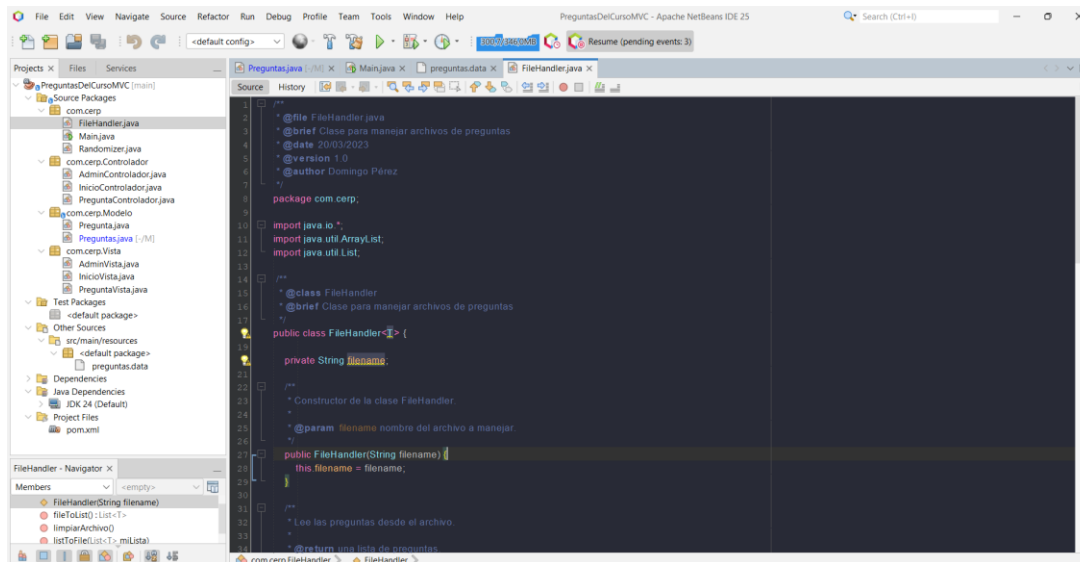
Fragmento de código Java de la clase FileHandler dentro del proyecto PreguntasDelCursoMVC, mostrando la declaración de la clase, la importación de librerías y el constructor que recibe el nombre del archivo.



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.



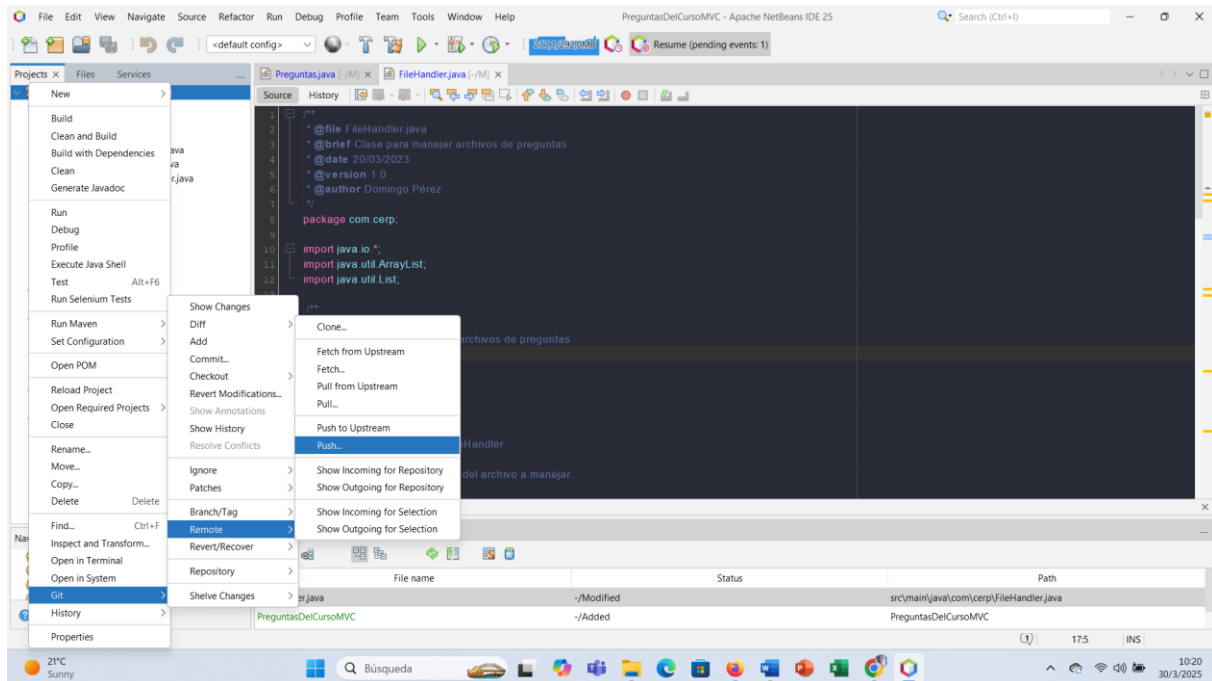
Nota. Elaboración propia.

Limpiar y reconstruir el proyecto en NetBeans:

Run > Clean and Build Project como se muestra en la siguiente imagen:

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

Pasos:

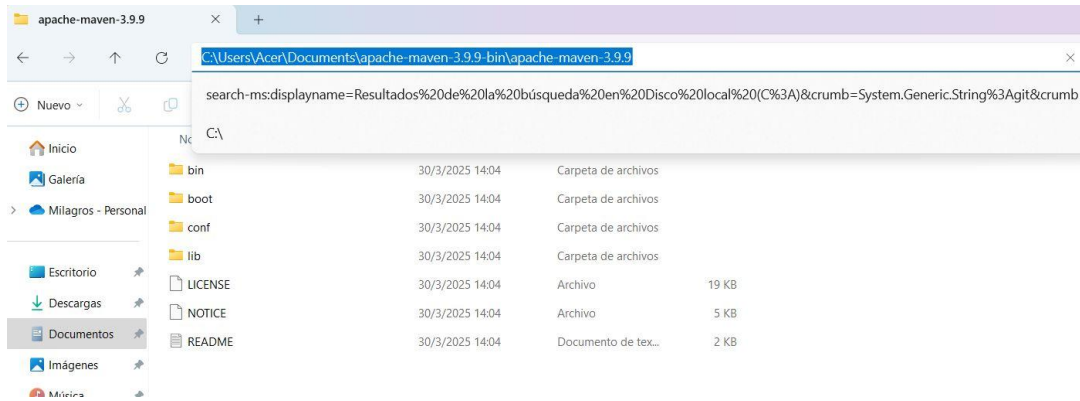
- 1-Descargar el archivo **bin.zip**.
- 2-Extraer la carpeta en C:\Program Files\Apache Maven.
- 3- Seguir los pasos de para agregarlo al PATH:

Buscar la carpeta donde está Maven, en mi caso:

C:\Users\Acer\Documents\apache-maven-3.9.9-bin\apache-maven-3.9.9 (ver Figura 14)

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

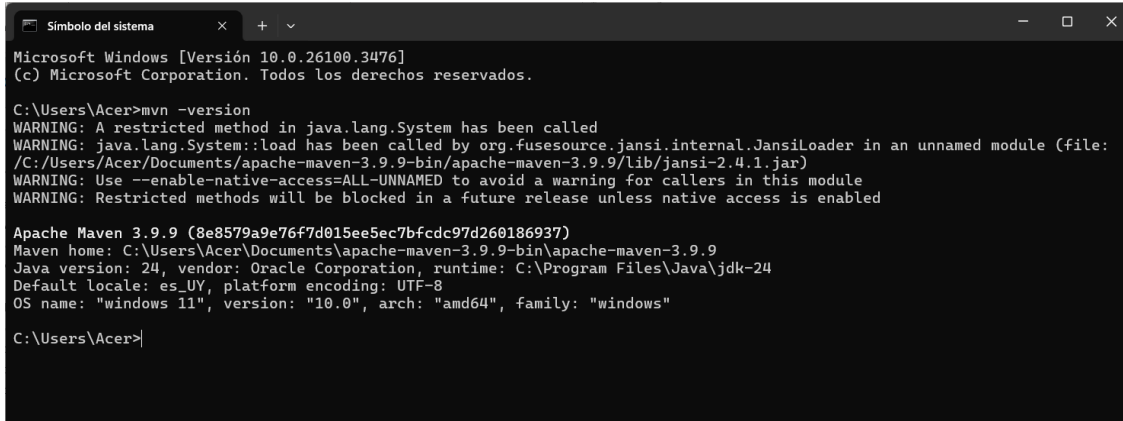
- Agregar la ruta de Maven a la variable de entorno **PATH**.
- Verificar la instalación desde la terminal con **mvn -version**.
- Pulsar Win + R, escribir **sysdm.cpl** y presionar Enter.
- Ir a **Opciones avanzadas** → **Variables de entorno**.
- En **Variables del sistema**, buscar Path y editarlo.
- 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, como se muestra en la siguiente imagen:

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.



```

Microsoft Windows [Versión 10.0.26100.3476]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Acer>mvn -version
WARNING: A restricted method in java.lang.System has been called
WARNING: java.lang.System::load has been called by org.fusesource.jansi.internal.JansiLoader in an unnamed module (file:
/C:/Users/Acer/Documents/apache-maven-3.9.9-bin/apache-maven-3.9.9/lib/jansi-2.4.1.jar)
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for callers in this module
WARNING: Restricted methods will be blocked in a future release unless native access is enabled

Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: C:\Users\Acer\Documents\apache-maven-3.9.9-bin\apache-maven-3.9.9
Java version: 24, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-24
Default locale: es_UY, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\Acer>

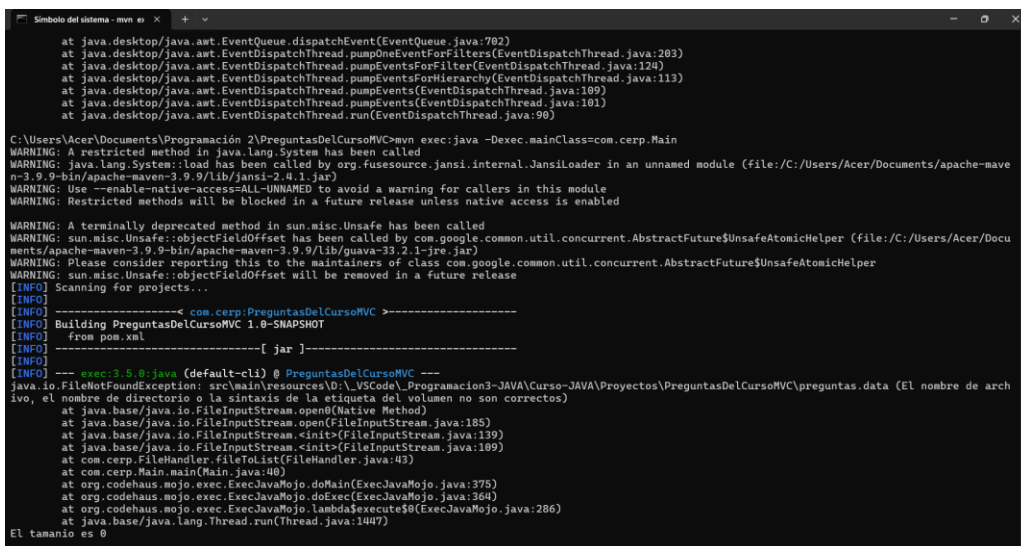
```

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 (ver Figura 16)

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.



```

at java.desktop/java.awt.EventQueue.dispatchEvent(EventQueue.java:782)
at java.desktop/java.awt.EventDispatchThread.pumpOneEventForFilters(EventDispatchThread.java:203)
at java.desktop/java.awt.EventDispatchThread.pumpEventsForFilter(EventDispatchThread.java:124)
at java.desktop/java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:113)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:109)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101)
at java.desktop/java.awt.EventDispatchThread.run(EventDispatchThread.java:90)

C:\Users\Acer\Documents\Programación 2\PreguntasDelCursoMVC>mvn exec:java -Dexec.mainClass=com.cerp.Main
WARNING: A restricted method in java.lang.System has been called
WARNING: java.lang.System::load has been called by org.fusesource.jansi.internal.JansiLoader in an unnamed module (file:/C:/Users/Acer/Documents/apache-maven-3.9.9-bin/apache-maven-3.9.9/lib/jansi-2.4.1.jar)
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for callers in this module
WARNING: Restricted methods will be blocked in a future release unless native access is enabled

WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by com.google.common.util.concurrent.AbstractFuture$UnsafeAtomicHelper (file:/C:/Users/Acer/Documents/apache-maven-3.9.9-bin/apache-maven-3.9.9/lib/guava-33.2.1-jre.jar)
WARNING: Please consider reporting this to the maintainers of class com.google.common.util.concurrent.AbstractFuture$UnsafeAtomicHelper
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
[INFO] Scanning for projects...
[INFO] -----[ com.cerp.PreguntasDelCursoMVC ]-----
[INFO] Building PreguntasDelCursoMVC 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO] -----[ exec:3.5.0:java (default-cli) @ PreguntasDelCursoMVC ]-----
java.io.FileNotFoundException: src/main/resources/D:\_VSCode\_Programacion3-JAVA\Curso-JAVA\Proyectos\PreguntasDelCursoMVC\preguntas.data (El nombre de archivo, el nombre de directorio o la sintaxis de la etiqueta del volumen no son correctos)
at java.base/java.io.FileInputStream.open0(Native Method)
at java.base/java.io.FileInputStream.open(FileInputStream.java:185)
at java.base/java.io.FileInputStream.<init>(FileInputStream.java:139)
at java.base/java.io.FileInputStream.<init>(FileInputStream.java:109)
at com.cerp.FileHandler.listFiles(FileHandler.java:43)
at com.cerp.Main.main(Main.java:40)
at org.codehaus.mojo.exec.ExecJavaMojo.doMain(ExecJavaMojo.java:375)
at org.codehaus.mojo.exec.ExecJavaMojo.doExec(ExecJavaMojo.java:364)
at org.codehaus.mojo.exec.ExecJavaMojo.lambda$execute$0(ExecJavaMojo.java:286)
at java.base/java.lang.Thread.run(Thread.java:1447)

El tamaño es 0

```

Nota. Elaboración propia

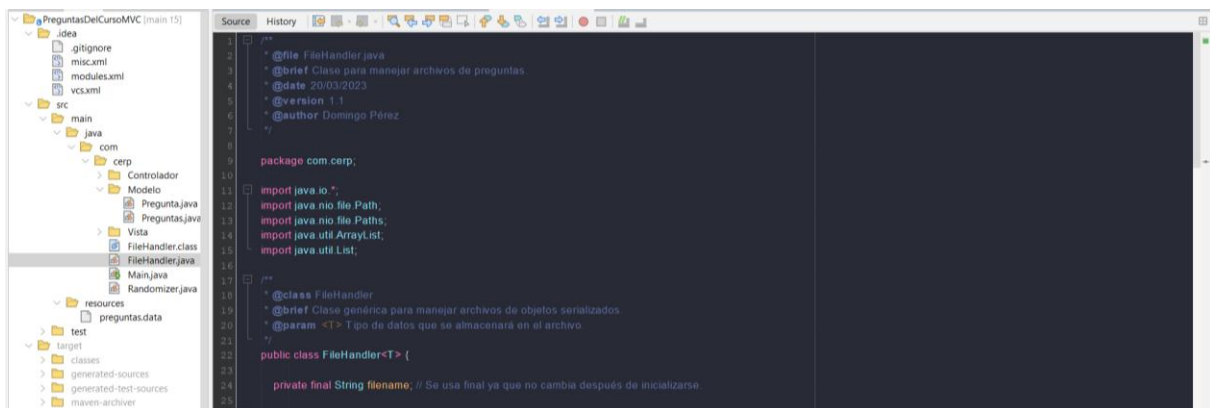


Corrección de Advertencias en la Documentación Javadoc

Se agregó la etiqueta **@param** en **FileHandler<T>** para documentar correctamente el tipo de datos genérico, como se muestra en la siguiente imagen:

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*.

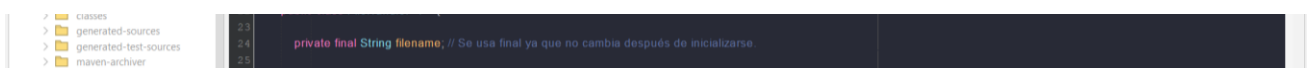


Nota. Elaboración propia

Además, se corrigió la advertencia sobre **filename**, declarándolo como **final** (ver Figura 18):

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.+Características+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+Introducción+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.+Características+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+Introducción+a+POO.pdf" se define: "Un objeto es una entidad que encapsula tanto datos como comportamientos".²
5. "¿Qué es la herencia en Java?"³ - Esta pregunta se relaciona directamente con el PDF "1.2+Conceptos+POO.+Características+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.*

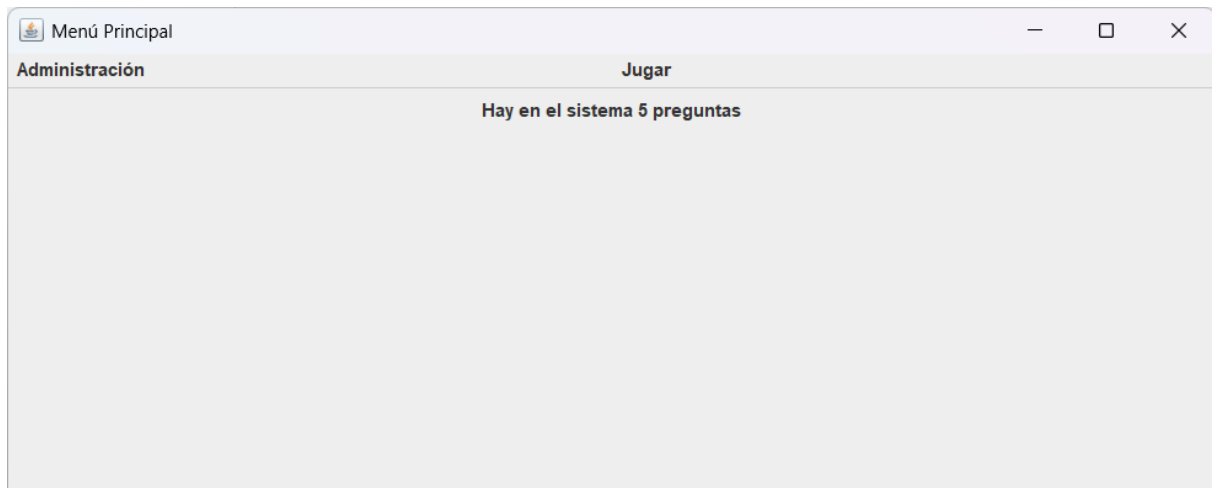
Se adjuntan a continuación imágenes mostrando el funcionamiento del proyecto:

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.

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

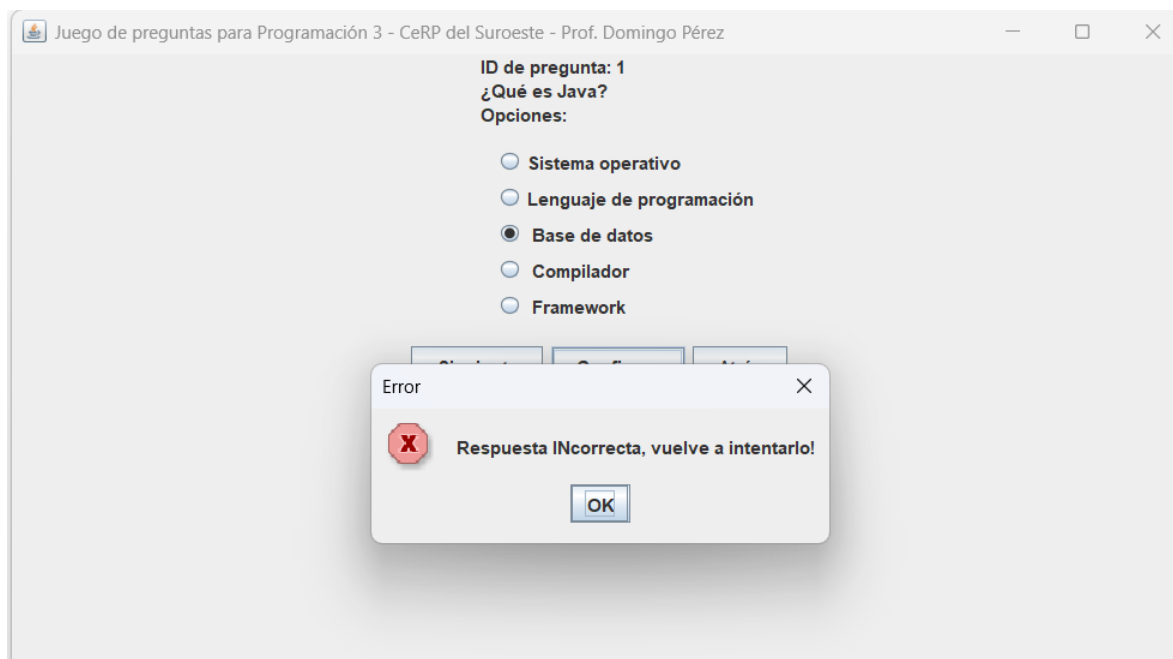
**Figura 19**

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:





Juego de preguntas para Programación 3 - CeRP del Suroeste - Prof. Domingo Pérez

ID de pregunta: 2
¿Qué es la Programación Orientada a Objetos?
Opciones:

- ☐ Un lenguaje de programación
- ☐ Una base de datos
- ☐ Un paradigma de programación
- ☐ Una metodología de diseño
- ☐ Un conjunto de reglas

Juego de preguntas para Programación 3 - CeRP del Suroeste - Prof. Domingo Pérez

ID de pregunta: 2
¿Qué es la Programación Orientada a Objetos?
Opciones:

- ☐ Un lenguaje de programación
- ☐ Una base de datos
- ☒ Un paradigma de programación
- ☐ Una metodología de diseño
- ☐ Un conjunto de reglas

Confirmación

Respuesta correctamente.

Juego de preguntas para Programación 3 - CeRP del Suroeste - Prof. Domingo Pérez

ID de pregunta: 2
¿Qué es la Programación Orientada a Objetos?
Opciones:

- ☐ Un lenguaje de programación
- ☐ Una base de datos
- ☐ Un paradigma de programación
- ☒ Una metodología de diseño
- ☐ Un conjunto de reglas

Error

Respuesta INcorrecta, vuelve a intentarlo!

Juego de preguntas para Programación 3 - CeRP del Suroeste - Prof. Domingo Pérez

ID de pregunta: 3
¿Qué es una clase en Java?

Opciones:

- ☐ Es un objeto
- ☐ Es una variable
- ☐ Es un método especial
- ☐ Es un tipo de dato
- ☐ Es un modelo o plantilla para crear objetos


Juego de preguntas para Programación 3 - CeRP del Suroeste - Prof. Domingo Pérez

ID de pregunta: 3
¿Qué es una clase en Java?

Opciones:

- ☐ Es un objeto
- ☒ Es una variable
- ☐ Es un método especial
- ☐ Es un tipo de dato
- ☐ Es un modelo o plantilla para crear objetos

Error

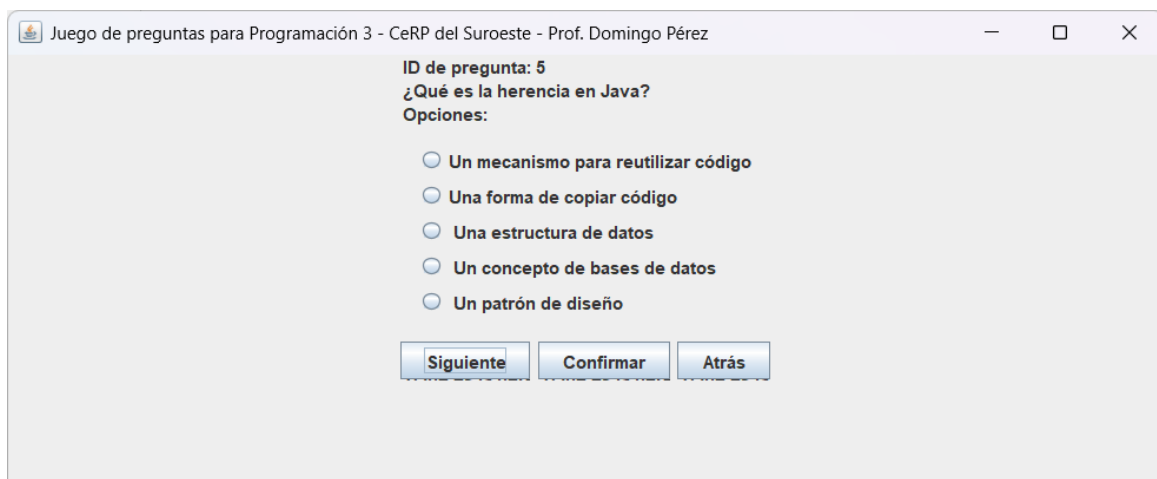
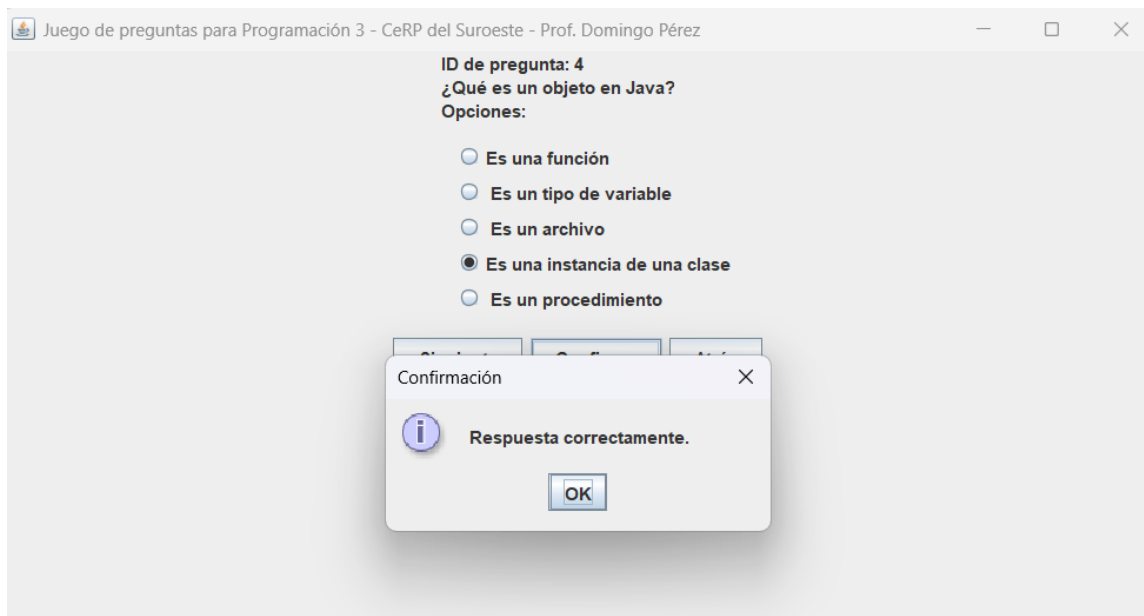
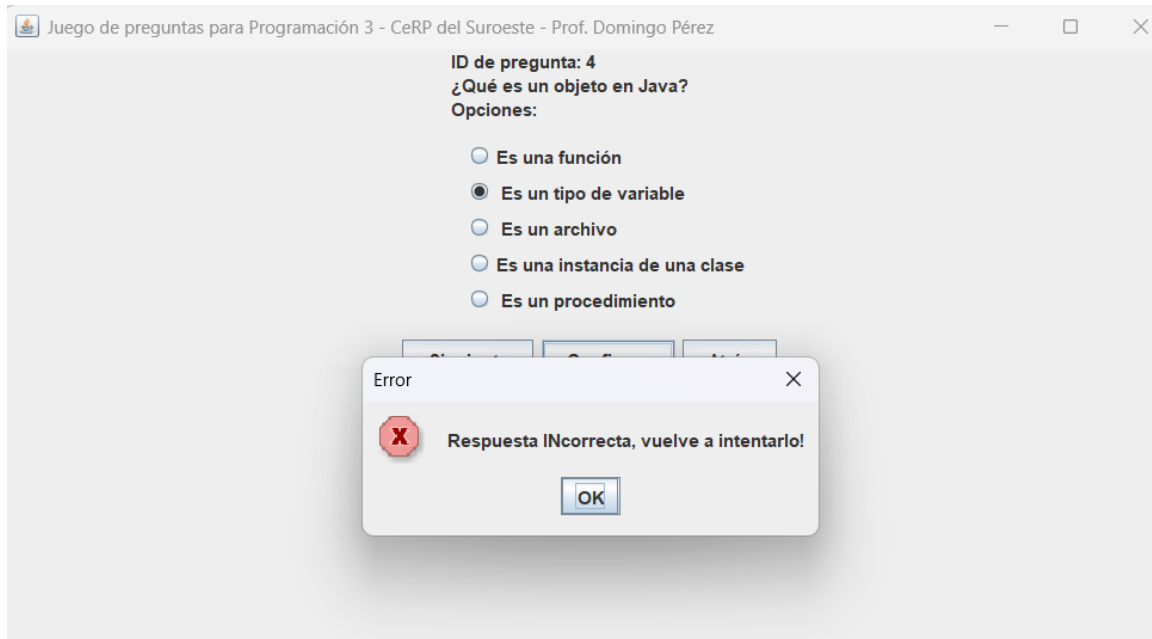
 Respuesta INcorrecta, vuelve a intentarlo!

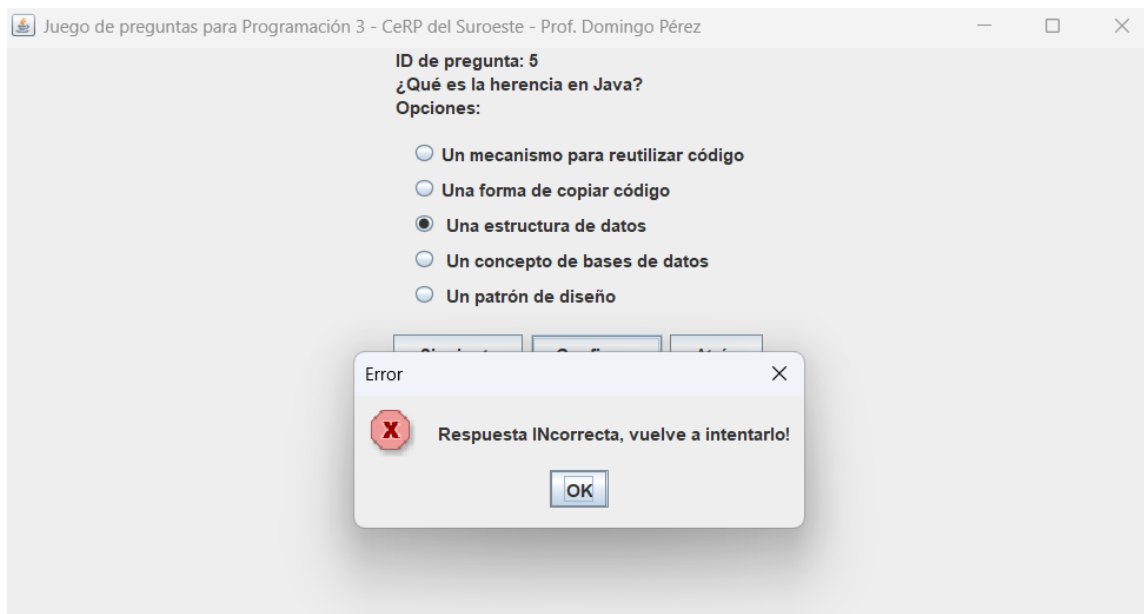
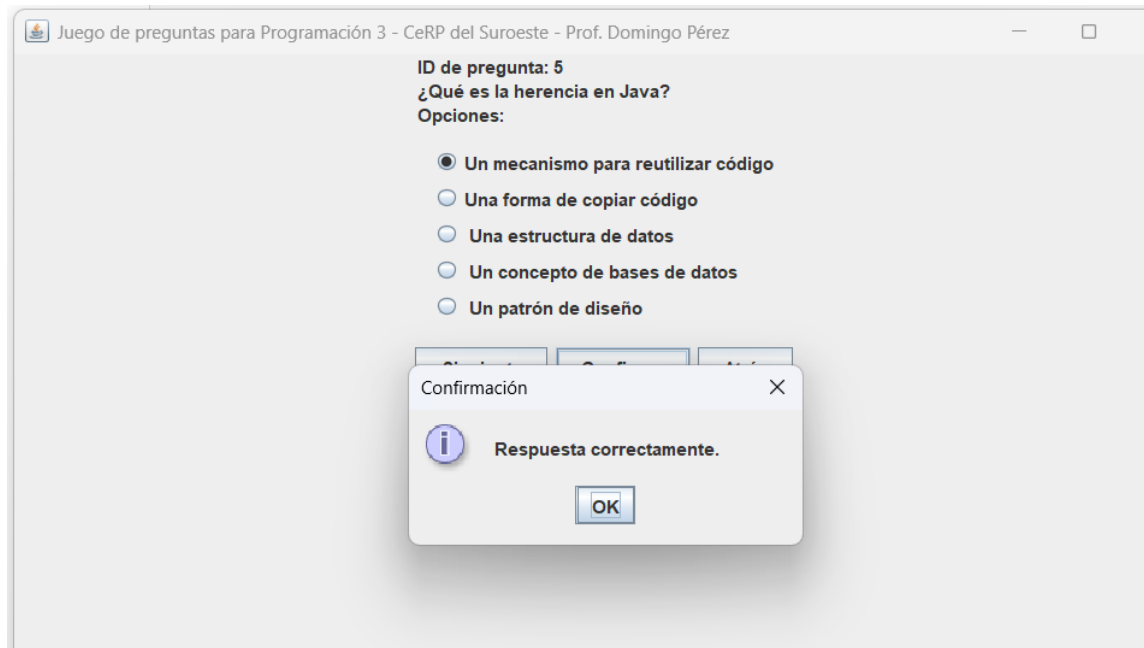
Juego de preguntas para Programación 3 - CeRP del Suroeste - Prof. Domingo Pérez

ID de pregunta: 4
¿Qué es un objeto en Java?

Opciones:

- ☐ Es una función
- ☐ Es un tipo de variable
- ☐ Es un archivo
- ☐ Es una instancia de una clase
- ☐ Es un procedimiento





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>

Liskov, B., & Wing, J. M. (1987). A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(6), 1811–1841.

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-Hill

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