**FUNDACION UNIVERSITARIA INTERNACIONAL**

**DE LA RIOJA COLOMBIA**

**ESPECIALIZACION EN INGENIERIA DE SOFTWARE**

**(PLATAFORMAS DE DESARROLLO DE SOFTWARE)**

**ACTIVIDAD 1**

**LABORATORIO DESARROLLO DE APLICACIONES EN NETBENS**

**HAROLD FERNANDO ROJAS LOPEZ**

**06 DICIEMBRE 2024**

# TABLA DE CONTENIDO

[TABLA DE CONTENIDO 1](#_Toc2014746246)

[INTRODUCCION. 2](#_Toc1734049869)

[DESARROLLO DE LA ACTIVIDAD. 3](#_Toc232144179)

[Tecnologías utilizadas. 3](#_Toc1714729484)

[Clases Java de la aplicación. 5](#_Toc1779176810)

[Procedimiento para la Toma de Pedidos 10](#_Toc497669468)

[Gestión de errores y notificación al usuario. 12](#_Toc69073072)

[Depuración NetBeans IDE. 14](#_Toc2041508183)

[CONCLUSIONES 15](#_Toc753335427)

# INTRODUCCION.

Este proyecto se llevó a cabo como parte de la materia "Plataformas de Desarrollo de Software", cuyo objetivo principal es aplicar los conocimientos adquiridos sobre entornos de desarrollo, específicamente en NetBeans. La actividad consistió en el desarrollo de una aplicación sencilla en Java para gestionar pedidos farmacéuticos. A través de este proyecto, se abordaron aspectos fundamentales como la creación de un proyecto en un entorno de desarrollo integrado (IDE), la implementación de interfaces gráficas de usuario (GUI), y el manejo de la depuración de errores. La aplicación desarrollada permite a los usuarios realizar pedidos de medicamentos, validar la entrada de datos, y gestionar errores de manera efectiva a través de mensajes informativos y alertas visuales.

# DESARROLLO DE LA ACTIVIDAD.

## Tecnologías utilizadas.

Las tecnologías utilizadas en el desarrollo del proyecto son las siguientes:

Apache NetBeans IDE 13: Apache NetBeans es un entorno de desarrollo integrado (IDE) de código abierto, utilizado para crear aplicaciones en diversos lenguajes, incluido Java. La versión 13 ofrece un conjunto completo de herramientas que simplifican la escritura, depuración y despliegue del código. Entre sus características más destacadas están:

* Editor avanzado con resaltado de sintaxis, autocompletado y refactorización.
* Integración con Maven y Git, facilitando la gestión de dependencias y control de versiones.
* Depurador visual, que permite identificar errores de manera interactiva. NetBeans fue clave en este proyecto para gestionar eficientemente la estructura del código y mejorar la productividad durante el desarrollo.

JDK 17:El Java Development Kit (JDK) 17 es la versión a largo plazo (LTS) del lenguaje Java. Proporciona mejoras en el rendimiento, nuevas características y mayor seguridad. Algunas de las ventajas más relevantes de esta versión incluyen:

* Mejoras en la sintaxis del lenguaje como las clases selladas y los patrones en expresiones switch, lo que permite escribir código más conciso y legible.
* Mayor rendimiento y soporte mejorado para la programación concurrente, clave para aplicaciones modernas.
* Compatibilidad con versiones anteriores, permitiendo la reutilización de bibliotecas y dependencias existentes. En este proyecto, JDK 17 permitió escribir código moderno y aprovechar sus nuevas características para construir una aplicación estable y eficiente.

Apache Maven:Apache Maven es una herramienta de gestión de proyectos y construcción automatizada. Fue utilizado para manejar las dependencias del proyecto y simplificar el proceso de compilación. Entre sus principales beneficios están:

* Gestión de dependencias: Maven permite integrar bibliotecas externas fácilmente.
* Construcción automatizada: Facilita la compilación del código y la generación de ejecutables en un entorno coherente.
* Estandarización del proyecto: A través de su estructura predefinida, garantiza un diseño uniforme y escalable. En este proyecto, Maven fue crucial para mantener una estructura ordenada y facilitar la integración con bibliotecas adicionales, como las proporcionadas por Java Swing.

Java Swing:Java Swing es una biblioteca de desarrollo de interfaces gráficas (GUI) que forma parte de la plataforma estándar de Java. Fue utilizada para crear una interfaz amigable y funcional para el usuario final. Algunas características clave de Swing son:

* Componentes reutilizables: Ofrece elementos como botones, campos de texto, etiquetas y tablas.
* Personalización y flexibilidad: Permite modificar el diseño y comportamiento de los componentes según las necesidades específicas del proyecto.
* Independencia del sistema operativo: Las aplicaciones desarrolladas con Swing funcionan en cualquier plataforma compatible con Java. Swing fue utilizado para construir formularios dinámicos e interactivos, que permitieron a los usuarios gestionar pedidos de forma intuitiva y eficiente.

## Clases Java de la aplicación.

A continuación, se listan las clases Java principales del aplicativo y el rol que cumplen dentro de este.

Clase PharmaOrderManager: la clase **PharmaOrderManager** desempeña el rol de punto de entrada principal del proyecto. Su función es inicializar y mostrar la ventana JFrame principal de la aplicación llamada **PharmaOrderForm,** configurándola para que se abra centrada en la pantalla y sea visible para el usuario.

Este archivo representa el núcleo inicial del proyecto, ya que:

* Contiene el método main, desde el cual se ejecuta la aplicación.
* Crea una instancia de la clase **PharmaOrderForm**, que gestiona la interfaz gráfica principal para la captura y gestión de pedidos farmacéuticos.

En resumen, **PharmaOrderManager** es el encargado de lanzar la aplicación y establecer su ventana principal como el punto de interacción inicial para los usuarios.

Clase PharmaOrderForm: La clase **PharmaOrderForm** es una interfaz gráfica de usuario (GUI) que extiende de javax.swing.JFrame y permite gestionar la toma de pedidos farmacéuticos en la aplicación **PharmaCare**. Su principal función es capturar la información del pedido ingresada por el usuario, validar los datos, y comunicarse con otras vistas como **OrderSummaryForm**.

* **Propiedades principales**:
  + *txtNameMedicine:* Campo de texto para el nombre del medicamento.
  + *cbxTypeMedicine:* ComboBox para seleccionar el tipo de medicamento (ej. analgésico, antibiótico, etc.).
  + *txtRequiredQuantity:* Campo de texto para ingresar la cantidad requerida del medicamento.
  + *btgDistributor:* Grupo de botones de radio que permite seleccionar un distribuidor farmacéutico (Cofarma, Empsephar, Cemefar).
  + *chbPrincipal, chbSecondary:* CheckBoxes para elegir las sucursales (principal o secundaria).
  + *btnConfirm, btnClean:* Botones para confirmar o limpiar el formulario.
* **Métodos principales:**
* *getSelectedDistributor():* Retorna el distribuidor seleccionado (Cofarma, Empsephar, o Cemefar) basado en el botón de radio activo.
* *setTextFieldMaxLength(JTextField textField, int maxLength):* Restringe el número máximo de caracteres que un usuario puede ingresar en un campo de texto, En este proyecto se restringe la longitud de caracteres en los campos de texto para: el nombre del medicamento (40 caracteres) y la cantidad requerida (7 caracteres).
* *resetLabelColors():* Restaura los colores de las etiquetas del formulario a su estado original (negro), útil después de realizar validaciones.
* *setComponentsEnabled(boolean enabled):* Activa o desactiva los campos y botones del formulario, útil cuando el pedido ya ha sido procesado.
* *cleanForm():* Limpia todos los campos del formulario y restaura sus valores predeterminados.
* *btnConfirmActionPerformed():* 
  + Valida los datos ingresados utilizando la clase PharmacyOrder.
  + Muestra mensajes de error en las etiquetas correspondientes si hay problemas de validación.
  + Si los datos son válidos:
    - Desactiva el formulario.
    - Pasa el pedido a OrderSummaryForm para mostrar el resumen.
    - Navega hacia la vista del resumen.
* *btnCleanActionPerformed():*
  + Solicita confirmación al usuario y limpia los datos del formulario si este acepta.

Clase PharmacyOrder: La clase **PharmacyOrder** representa un modelo de datos en la aplicación **PharmaCare** que encapsula la información relacionada con un pedido farmacéutico. Su principal responsabilidad es almacenar los datos del pedido y proporcionar métodos para validar su consistencia y formato.

* **Propiedades principales:**
  + *nameMedicine*: Nombre del medicamento solicitado.
  + *typeMedicine*: Identificador numérico del tipo de medicamento (ej. 1 = analgésico, 2 = antibiótico).
  + *medicineTypeName*: Nombre descriptivo del tipo de medicamento.
  + *requiredQuantity*: Cantidad solicitada como cadena para facilitar la entrada y validación.
  + *distributor*: Distribuidor farmacéutico seleccionado (Cofarma, Empsephar, Cemefar).
  + *isPrincipalPharmacy* y *isSecondaryPharmacy*: Indicadores booleanos que identifican si el pedido está destinado a la farmacia principal, secundaria o ambas.
  + *isValid*: Booleano que indica si el pedido es válido después de realizar las validaciones.
  + *validationMessages:* Mensajes acumulativos que detallan los errores de validación.
  + *validNameMedicine, validTypeMedicine, validRequiredQuantity, validDistributor, validSubsidiary:* Booleanos específicos que indican si cada campo cumple con los criterios de validación.
* **Métodos principales:**
* getAddressInfo():
  + - Genera una cadena de texto que describe la(s) dirección(es) asociada(s) al pedido, basándose en si está destinado a la farmacia principal, secundaria o ambas.
    - Ejemplo de salida: "Para la farmacia situada en Calle de la Rosa n.28 y para la situada en Calle Alcazabilla n.3".
* *validate():* 
  + - Valida los datos ingresados en los atributos de la clase.
    - Genera mensajes de error acumulativos en validationMessages si los datos son inválidos.
    - Realiza las siguientes validaciones:
      * El nombre del medicamento debe ser alfanumérico y no estar vacío.
      * El tipo de medicamento debe estar seleccionado.
      * La cantidad solicitada debe ser un número entero positivo.
      * Debe seleccionarse un distribuidor.
      * Debe seleccionarse al menos una farmacia (principal o secundaria).
* Retorna un booleano indicando si todos los datos son válidos (true) o si hay errores (false).
* Actualiza los campos de validación individuales (ej. validNameMedicine) y el estado general (isValid).

La clase **PharmacyOrder** desempeña un papel fundamental en la arquitectura de la aplicación, sirviendo como un modelo para almacenar los datos relacionados con un pedido farmacéutico. Sus métodos de validación garantizan la consistencia de los datos antes de ser procesados o almacenados. Además, su diseño facilita la detección de errores y permite generar respuestas claras para el usuario, mejorando la experiencia y confiabilidad del sistema.

Clase OrderSummaryForm: La clase **OrderSummaryForm** es una ventana gráfica en la aplicación PharmaCare diseñada para mostrar un resumen de los detalles de un pedido farmacéutico. Permite a los usuarios visualizar la información del pedido antes de enviarlo o cancelarlo. Se conecta con otras clases del sistema como **PharmacyOrder** (modelo de datos) y **PharmaOrderForm** (formulario principal).

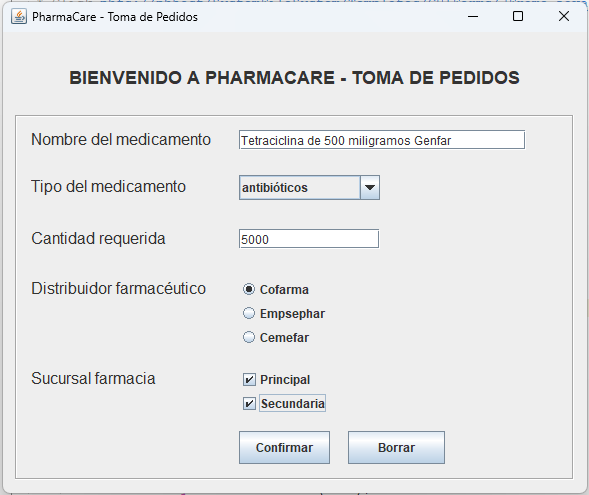
* **Propiedades principales:**
  + *PharmacyOrder order*: Objeto que contiene los datos del pedido farmacéutico, como medicamento, cantidad, distribuidor y destino.
  + *PharmaOrderForm mainForm:* Referencia al formulario principal (PharmaOrderForm) para coordinar la navegación y la gestión del estado de los componentes gráficos.
  + *Componentes de la interfaz gráfica:*
    - *lblMedicineInfo:* Etiqueta para mostrar información del medicamento y la cantidad solicitada.
    - *lblAddressInfo:* Etiqueta que muestra la dirección de envío del pedido.
    - *btnEnviar:* Botón para enviar el pedido.
    - *btnCancel:* Botón para cancelar el pedido.
* **Metodos princiaples:**
  + *updateLabels():* 
    - Actualiza las etiquetas de la ventana gráfica con la información del pedido (order).
    - Formatea los detalles como el nombre del medicamento, la cantidad y la dirección de envío utilizando los datos del pedido.
    - Cambia el título de la ventana según el distribuidor seleccionado.
* btnEnviarActionPerformed():
  + Ejecutado al presionar el botón "Enviar pedido".
  + Muestra un mensaje de éxito indicando que el pedido ha sido enviado correctamente.
  + Limpia las etiquetas de la ventana y cierra la ventana actual.
  + Reactiva el formulario principal (mainForm) y lo deja listo para nuevos pedidos.
* btnCancelActionPerformed():
  + Ejecutado al presionar el botón "Cancelar pedido".
  + Solicita confirmación al usuario antes de cancelar el pedido.
  + Si se confirma, limpia los campos de la ventana, la cierra y reactiva el formulario principal (mainForm).
* setOrder(PharmacyOrder order):
  + Asigna un objeto PharmacyOrder a la ventana para que pueda mostrar sus datos en las etiquetas.
* setMainForm(PharmaOrderForm mainForm):
  + Establece una referencia al formulario principal para permitir la comunicación bidireccional.

La clase **OrderSummaryForm** actúa como un puente entre el usuario y los datos del pedido farmacéutico. Sus principales funciones son:

* Proporcionar una vista clara y organizada del pedido para su revisión.
* Permitir al usuario tomar decisiones sobre el envío o cancelación del pedido.
* Gestionar la interacción entre las interfaces gráficas, garantizando un flujo fluido entre las ventanas del sistema.

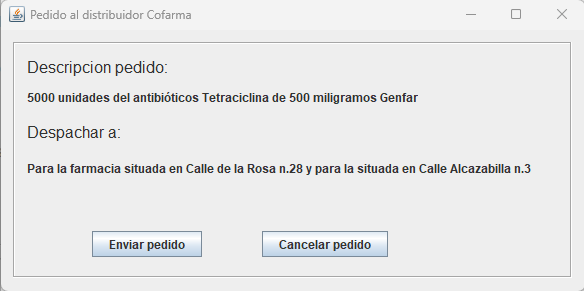
## Procedimiento para la Toma de Pedidos

En este ejemplo, realizaremos la toma de un pedido de cinco mil unidades del medicamento Tetraciclina de 500 miligramos, clasificado como antibiótico, fabricado por el laboratorio Genfar. Este pedido será gestionado a través del distribuidor Cofarma y estará destinado a las sucursales Principal y Secundaria.



*Figura 1. Pantalla principal para la toma de los datos del pedido.*

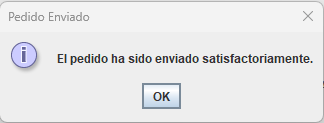
Una vez confirmado el pedido y verificado que los datos ingresados sean correctos, se mostrará una segunda pantalla con el detalle completo del pedido.



*Figura 2. Pantalla secundaria donde se muestra el detalle del pedido.*

En la pantalla de detalle del pedido, el título muestra el nombre del distribuidor al que se realizará la solicitud, en este caso, “**Cofarma**”. A continuación, se presenta la descripción del pedido, que incluye el número de unidades requeridas, el tipo de medicamento y su nombre. En la sección “**Despachar a**”, se especifican las direcciones de envío para ambas farmacias: la principal y la secundaria.

Finalmente, al presionar el botón “Enviar pedido”, se notificará al usuario mediante un mensaje que el pedido ha sido enviado con éxito. Además, el mismo mensaje se registrará en la consola para referencia.



*Figura 3. Mensaje que informa al usuario que el pedido se ha enviado satisfactoriamente.*



*Figura 4. Impresión del mensaje por consola.*

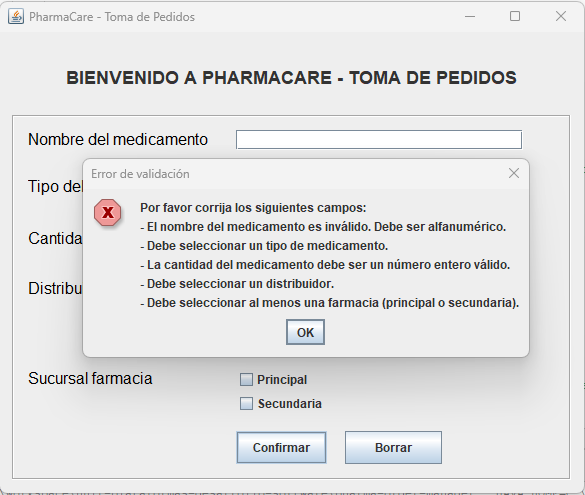
## Gestión de errores y notificación al usuario.

El sistema gestiona los errores mediante un proceso de validación implementado en la clase “**PharmacyOrder**” en el método “**validate**”. Cada vez que el usuario intenta enviar un pedido, los datos ingresados se evalúan en esta clase para garantizar que cumplan con los requisitos definidos. Durante este proceso, se valida que todos los campos obligatorios estén completos, que los valores ingresados sean correctos y que las selecciones sean válidas.

Si se detectan errores, el sistema marca automáticamente los campos inválidos en la interfaz. Esto se logra actualizando a color rojo los JLabel asociados a dichos campos. Por ejemplo, si el usuario no ingresa un nombre válido para el medicamento, el sistema mostrará un mensaje como "El nombre del medicamento es inválido. Debe ser alfanumérico" y se pondrá en rojo el JLabel asociado a ese campo.

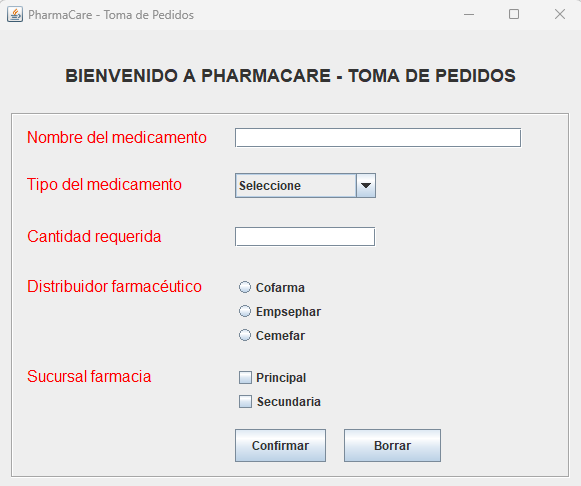
Además, se acumulan todos los mensajes de error en una lista para que el usuario pueda ver de forma integral todos los problemas que necesitan ser corregidos antes de proceder. Solo cuando todos los campos han sido corregidos y validados con éxito, se permite al usuario avanzar al siguiente paso en el proceso de toma de pedido. Esto garantiza una experiencia clara y evita que se envíen pedidos con información incompleta o incorrecta.

A continuación, un ejemplo donde todos los campos del pedido son incorrectos.



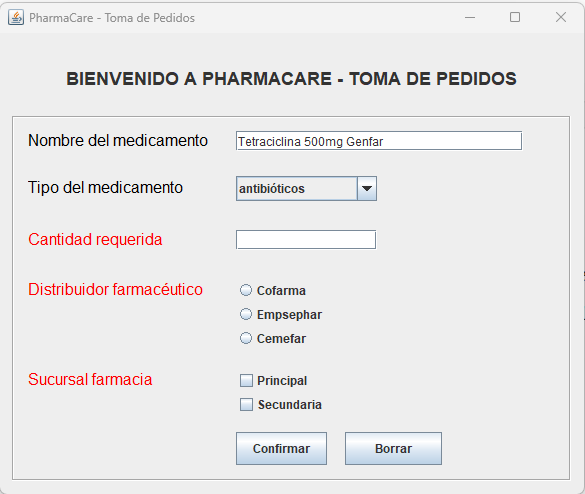
*Figura 5. Mensaje de error detallado de los campos que no pasaron la validación.*

Cuando el sistema detecta campos incorrectos, muestra un mensaje de alerta con un listado detallado de todos los campos que requieren atención. Al presionar el botón “OK”, los “JLabel” asociados a esos campos se resaltan en rojo, indicando visualmente al usuario cuáles necesitan ser corregidos.



*Figura 6. Labels en color rojo indicando los campos que presentan error.*

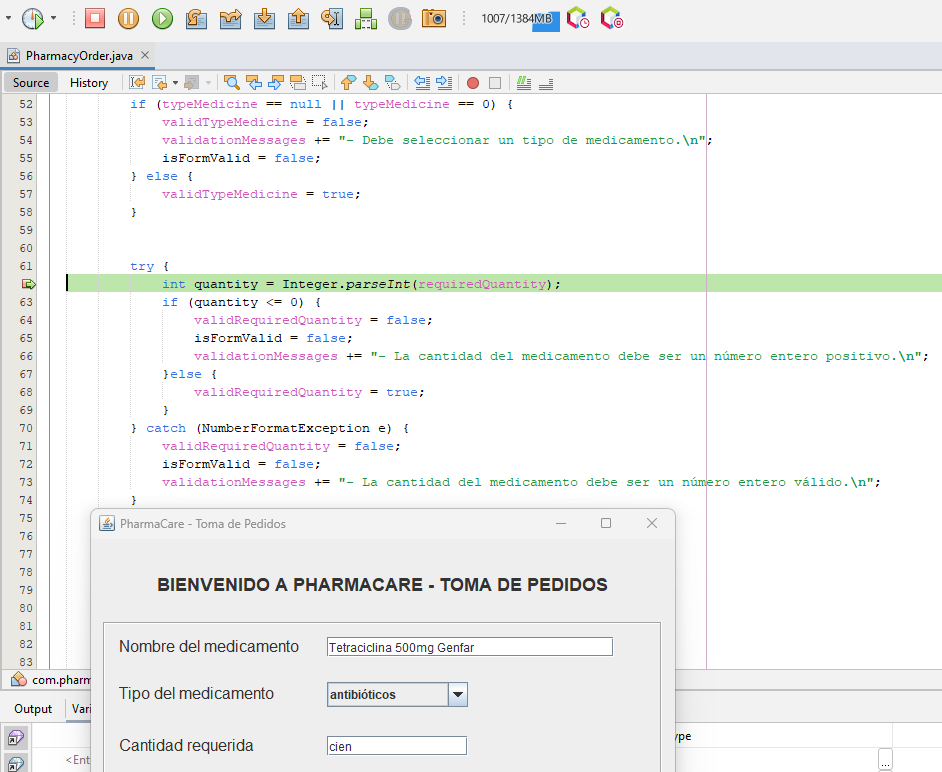
A medida que se corrigen los campos con errores y se presiona nuevamente el botón Confirmar, los JLabel asociados recuperan su color original (Negro), indicando que los datos ingresados son ahora válidos.



*Figura 7. Campos que han pasado la validación correctamente.*

## Depuración NetBeans IDE.

Se coloca un “breakpoint” en la sección donde se realiza la validación del campo “cantidad requerida”, en el método “validate” de la clase “PharmacyOrder”. Esta validación intenta convertir el valor ingresado en el campo “requiredQuantity” a un número entero. Si el valor no es un número entero positivo, o si no se puede convertir debido a un error de formato, se marca el campo como inválido y se agrega un mensaje de error. Si el valor es correcto, el campo se valida como correcto.



*Figura 8. Punto de interrupción y depuración en el IDE Netbeans.*

# CONCLUSIONES

1. Aprendizaje de herramientas y entornos de desarrollo: El proyecto permitió familiarizarme con el uso del IDE NetBeans, una herramienta ampliamente utilizada para el desarrollo en Java. A través de la práctica, se comprendió cómo crear y gestionar proyectos, además de aprender a utilizar las funcionalidades del IDE para facilitar la depuración de errores, una habilidad clave en el proceso de desarrollo.

2. Desarrollo de la interfaz gráfica: El diseño y la implementación de la interfaz gráfica de usuario fueron fundamentales para la interacción con el usuario. Mediante el uso de componentes como formularios, etiquetas y botones, se pudo proporcionar una experiencia de usuario fluida y clara. La capacidad de mostrar mensajes de error y validaciones en tiempo real también fue crucial para garantizar la usabilidad del sistema.

3. Manejo de errores y validación de datos: Uno de los aspectos más destacados del proyecto fue la implementación de validaciones para asegurar que los datos ingresados fueran correctos antes de enviarlos. Se aprendió a gestionar y mostrar mensajes de error de manera adecuada, mejorando la fiabilidad y la robustez de la aplicación. Este aspecto también incluyó el uso de breakpoints para depurar y validar la lógica de la aplicación en tiempo de ejecución.

4. Mejora de habilidades en programación Java: A lo largo del proyecto, se reforzaron los conocimientos de programación en Java, específicamente en lo relacionado con la manipulación de formularios, el manejo de excepciones, y la interacción con el usuario a través de interfaces gráficas. La experiencia adquirida será útil para proyectos más complejos en el futuro.