Documentación de API Java

12 Clases | 0 Interfaces

Diagrama de Clases

Diagrama generado automáticamente del análisis del código Java

Clases

LoginController

LoginController es un controlador que maneja la autenticación de usuarios, el registro, obtención del nombre de usuario y creación de tareas.

Métodos:

- **LoginController**(void): public — Clase: LoginController
- **login**(String, String): boolean — El método `login` de la clase `LoginController` verifica las credenciales de acceso de un usuario para determinar si el login es válido. Devuelve un valor booleano (`true` si el login es exitoso, `false` en caso contrario). Su propósito principal es autenticar al usuario mediante una combinación de nombre de usuario y contraseña, generalmente validando estos datos contra una base de datos o servicio de autenticación. Es un componente clave del flujo de inicio de sesión en una aplicación web o móvil, garantizando el control de acceso a recursos protegidos.
- **register**(String, String, String): boolean — El método `register` de la clase `LoginController` permite registrar un nuevo usuario en el sistema. Devuelve un valor de tipo `boolean`, donde `true` indica que el registro se realizó exitosamente y `false` sugiere que ocurrió un error o que las condiciones de registro no se cumplieron (por ejemplo, datos inválidos o

duplicación de usuario). Su propósito principal es gestionar la lógica de creación de cuentas, validando los datos proporcionados (como correo electrónico o contraseñas) y asegurándose de que el usuario no ya exista en la base de datos. Es un componente clave del flujo de autenticación, permitiendo a los usuarios crear sus propias cuentas mediante una interfaz de controlador.

- **getUsername**(void): String
- **createTask**(String): void
- **launchMainView**(void): void
- **launchAddTask**(void): void

TasksController

Controlador de tareas que proporciona métodos para obtener y gestionar las tareas desde una base de datos y establecer la vista principal.

Métodos:

- **TasksController**(void): private — La clase TasksController no está definida en el código proporcionado y no se especifica un método que devuelva privado. Por lo tanto, no es posible ofrecer una descripción técnica precisa de su función o propósito. Se requiere más información sobre la implementación real de la clase (por ejemplo, sus métodos, sus propiedades, su estructura de controladores y su relación con el manejo de tareas). Actualmente, dado que el código no está disponible ni se proporciona suficiente contexto, se considera que esta solicitud carece de datos técnicos relevantes para una análisis válido.
- **getInstance**(void): TasksController — La línea de código `TasksController.getInstance` devuelve una instancia única del controlador `TasksController`. Su propósito es proporcionar un acceso global y compartido a la clase `TasksController`, asegurando que solo se cree una instancia de esta clase durante toda la ejecución del programa. Esto es típico en patrones de diseño como el "Singleton", lo cual garantiza que las operaciones relacionadas con la gestión de tareas (como crear, actualizar o eliminar tareas) sean eficientes y consistentes, evitando la creación redundante de objetos. Es importante destacar que este método asume que `getInstance` devuelve una instancia ya existente o crea una

nueva si no existe anteriormente, lo cual garantiza el acceso controlado a los recursos del controlador.

- **setMainView**(MainView): void — El método setMainView de la clase TasksController establece la vista principal del sistema. Su propósito es configurar o asignar la interfaz gráfica principal (vista) que será utilizada para mostrar y gestionar las tareas. Como el método no devuelve ningún valor (void), actúa exclusivamente como un setter que modifica el estado interno de la clase al establecer una referencia a la vista principal, sin generar un resultado o retorno. Esta acción es fundamental para iniciar o actualizar la presentación del módulo de tareas en la aplicación.
- **loadTasksFromDatabase**(void): void
- **getTasks**(boolean, boolean): List
- **reloadTasksInMainView**(void): void
- **deleteTask**(int): void
- **CreateTask**(String, String, String): void
- **setTaskAsFinished**(int): void
- **login**(String, String): boolean
- *(+1 métodos más)*

Launch

Clase Launch que contiene métodos para determinar si es el primer lanzamiento, establecer si se mostró la pantalla de carga y ejecutar el punto principal.

Métodos:

- **main**(String[]): void — El método main de la clase Launch sirve como punto de entrada principal de una aplicación en Java. Su propósito es iniciar el programa al ejecutarlo desde la línea de comandos. Aunque este método no devuelve ningún valor (se especifica como void), realiza las tareas iniciales necesarias para lanzar la ejecución del sistema, como crear instancias de objetos o inicializar componentes clave. Es importante destacar que, en un entorno típico, el método main debe estar presente en cada clase principal de una aplicación Java y es el primer punto que se ejecuta al iniciar el programa.

- **isFirstLaunch**(void): boolean — El código `Launch.isFirstLaunch` es un método perteneciente a la clase `Launch` que devuelve un valor de tipo `boolean`. Su propósito es determinar si es la primera vez que se ejecuta la aplicación o el proceso de lanzamiento. Si el método retorna `true`, indica que el sistema ha sido iniciado por primera vez y no ha habido una ejecución previa; en caso contrario, retorna `false` cuando ya se ha realizado al menos una ejecución anterior. Este mecanismo es útil para mostrar pantallas de bienvenida, realizar configuraciones iniciales o evitar procesos repetitivos durante la ejecución del software.
- **setSplashShown**(void): void — El método `setSplashShown` de la clase `Launch` establece si se mostrará o no la pantalla de carga (splash screen) al iniciar la aplicación. Devuelve un valor void, lo que indica que el método no devuelve ningún valor. Su propósito principal es controlar visualmente la experiencia inicial del usuario, permitiendo activar o desactivar la presentación de la interfaz de carga. Esta característica puede ser útil en entornos de desarrollo para evitar mostrar pantallas de carga durante pruebas o en versiones sin interfaz gráfica. Es un método de configuración que afecta el comportamiento inicial del lanzador de la aplicación.

AppState

AppState es una clase que contiene un método principal y funciones para determinar si es la primera ejecución y para marcar que la pantalla de carga ha sido mostrada.

Métodos:

- **main**(String[]): void — El método main de la clase AppState inicializa la aplicación y actúa como punto de entrada principal del programa. Su propósito es ejecutar el código necesario para iniciar la aplicación, cargar recursos, configurar el estado inicial y comenzar el ciclo principal de ejecución. Este método no devuelve ningún valor (void), lo que indica que su función está orientada a realizar acciones de inicio en lugar de proporcionar un resultado computacional. Es fundamental para el funcionamiento del sistema, ya que sin él la aplicación no se inicia.
- **isFirstLaunch**(void): boolean — La propiedad `AppState.isFirstLaunch` es un método o campo de la clase `AppState` que devuelve un valor booleano indicando si el aplicativo ha sido lanzado por primera vez. Su

propósito principal es determinar si el sistema debe mostrar pantallas de bienvenida, realizar configuraciones iniciales o ejecutar procesos de inicio específicos. Si el valor es `true`, significa que la aplicación se está ejecutando en una sesión inicial; si es `false`, indica que ya ha sido lanzada anteriormente y los datos o configuraciones previas están disponibles. Esta lógica ayuda a optimizar el rendimiento y mejorar la experiencia del usuario al evitar procesos repetitivos en sesiones posteriores.

- **setSplashShown**(void): void — La línea de código `AppState.setSplashShown` es un método perteneciente a la clase `AppState` que establece el estado indicando si se ha mostrado o no la pantalla de carga (splash screen). Este método recibe como parámetro un valor booleano (por ejemplo, `true` para indicar que ya fue mostrada y `false` en caso contrario), y actualiza internamente el estado del objeto `AppState` para rastrear si la pantalla de inicio ha sido exhibida previamente. Su propósito principal es mantener un estado global que permite evitar mostrar repetidamente la pantalla de carga al iniciar la aplicación, mejorando así el rendimiento y la experiencia de usuario. Es un método de tipo void, lo cual indica que no devuelve ningún valor, pero sí modifica el estado interno del objeto.

Task

La clase Task representa una tarea con sus identificadores, usuario asociado, nombre, tipo y fecha de vencimiento.

Métodos:

- **getId**(void): int — El método getId de la clase Task devuelve un valor de tipo int que representa el identificador único de una tarea. Su propósito es proporcionar un número entero distinto para cada instancia de la clase Task, permitiendo así diferenciar entre tareas distintas en un sistema o conjunto de datos. Este identificador puede utilizarse para acceder a datos específicos, realizar búsquedas, o mantener referencias entre diferentes componentes del sistema. Es un método accesible y estándar que permite obtener el ID de la tarea sin necesidad de argumentos adicionales.
- **getUserId**(void): int — El método getUserId de la clase Task devuelve un valor de tipo int. Su propósito es obtener el identificador único asociado a

un usuario que está relacionado con una tarea. Este método permite acceder al ID del usuario responsables o vinculados a la tarea específica, facilitando operaciones como el seguimiento de responsabilidades, autenticación o gestión de permisos en el sistema. Es un acceso directo y eficiente para recuperar datos de identificación del usuario dentro del contexto de una tarea.

- **getName**(void): String — El método `getName()` de la clase `Task` devuelve un valor de tipo `String` que representa el nombre asignado a la tarea. Su propósito es proporcionar acceso al nombre de la instancia actual de la clase `Task`, lo cual puede ser útil para identificar o representar visualmente la tarea en interfaces de usuario, registros o operaciones de manejo de tareas. Es un método accesador (getter) simple y estático o instanciable, dependiendo del diseño de la clase, que no realiza operaciones complejas ni modifica el estado interno de la instancia. Esta funcionalidad es fundamental para la representación clara y directa de cada tarea en aplicaciones que requieren organización o visualización estructurada.
- **getTaskType**(void): String
- **getDueDate**(void): String
- **getRecurrencePattern**(void): String
- **isCompleted**(void): boolean
- **getCreatedAt**(void): LocalDateTime
- **isBacklog**(void): boolean
- **toString**(void): String
- *(+1 métodos más)*

User

La clase User representa a un usuario con nombre de usuario y contraseña, permitiendo obtener y modificar ambos atributos.

Métodos:

- **User**(String, String): public — La clase `User` define un modelo para representar a un usuario en una aplicación. El método público declarado en

esta clase permite acceder a atributos o funcionalidades relacionadas con el usuario, como su nombre, identificador o estado de autenticación. Su propósito principal es proporcionar una interfaz estándar para interactuar con la entidad "usuario", facilitando la gestión y recuperación de datos del usuario mediante operaciones públicas accesibles desde otros componentes del sistema. No se especifica el método exacto, pero su naturaleza pública indica que está diseñado para ser usado externamente, lo cual implica una buena separación de responsabilidades y acceso controlado a la información del usuario.

- **getUsername**(void): String — El método `getUsername()` de la clase `User` devuelve una cadena de caracteres que representa el nombre de usuario asociado al objeto `User`. Su propósito es obtener y acceder al valor del nombre de usuario de forma segura y estándar, permitiendo su uso en operaciones como autenticación o visualización de datos. Es un método de acceso público, no modificador, y asume que el campo correspondiente (por ejemplo, `username`) está correctamente inicializado antes de ser llamado. No realiza ninguna operación de cálculo ni validación adicional; su funcionalidad es simple y directa: obtener el valor almacenado del nombre de usuario.
- **getPassword**(void): String — El método `getPassword` de la clase `User` devuelve una cadena de caracteres que representa la contraseña del usuario. Su propósito es obtener el valor almacenado de la contraseña para su uso en operaciones de autenticación o verificación de credenciales. Es importante destacar que, por razones de seguridad, esta información no debería ser expuesta directamente en el código o en interfaces públicas, ya que podría comprometer la integridad y confidencialidad del sistema. Si se requiere acceso a la contraseña, debe hacerse mediante mecanismos seguros como hashing o autenticación basada en tokens, y nunca debe retornarse en texto plano.
- **setUsername**(String): void
- **setPassword**(String): void

DatabaseService

DatabaseService es una clase que proporciona métodos para obtener credenciales de usuario y contraseña, establecer conexión con la base de datos,

verificar si una base de datos existe y ejecutar consultas devolviendo el número de filas afectadas.

Métodos:

- **getUsername**(void): String — El método getUsername de la clase DatabaseService se encarga de obtener el nombre de usuario desde una base de datos. Su propósito es recuperar un valor de texto (cadena) asociado al nombre de usuario de un usuario específico, generalmente mediante una consulta SQL a la base de datos. Este método es fundamental para autenticación o para acceder a información de perfil del usuario en aplicaciones que dependen de una fuente de datos centralizada. Se asume que recibe parámetros necesarios (como el ID de usuario) y devuelve un String correspondiente al nombre de usuario almacenado en la base de datos. Es un método clave dentro de la capa de servicio para integrar funcionalidades de autenticación con el sistema de gestión de datos.
- **getPassword**(void): String — El método getPassword de la clase DatabaseService devuelve una cadena de caracteres que representa la contraseña utilizada para autenticar acceso a una base de datos. Su propósito principal es proporcionar el valor de contraseña necesitado por otros componentes del sistema para establecer conexiones seguras con la base de datos. Es importante destacar que este método debe usarse con precaución debido al riesgo de exposición de credenciales sensibles; en entornos productivos, se recomienda almacenar las contraseñas de forma cifrada o mediante gestores de secretos y no exponerlas directamente en el código.
- **connect**(void): Connection — El método `connect` de la clase `DatabaseService` establece una conexión con una base de datos. Su propósito es inicializar y devolver un objeto de tipo `Connection` que permite realizar operaciones como consultas, inserciones o actualizaciones en la base de datos. Este método generalmente utiliza información de configuración (como URL, usuario, contraseña) para autenticarse y establecer una conexión mediante un driver JDBC. Es fundamental que sea invocado antes de ejecutar cualquier operación directa sobre los datos. La conexión debe gestionarse adecuadamente para evitar pérdidas de recursos, como cerrarla al finalizar el uso (por ejemplo, usando try-with-resources o patrones de manejo de conexiones).

- **databaseExists**(void): boolean
- **executeQuery**(String): int
- **updateTask**(Task): int
- **login**(String, String): boolean
- **register**(String, String, String): boolean
- **getTasksByQuery**(String): List
- **extractTaskFromResultSet**(ResultSet): Task
- *(+10 métodos más)*

UIStyleManager

UIStyleManager es una clase que gestiona el estilo de la interfaz gráfica, aplicando temas oscuros, cargando fuentes, añadiendo marcos redondeados y gestionando eventos de tamaño y dibujo de imágenes.

Métodos:

- **applyDarkTheme**(void): void — La metodología `applyDarkTheme` de la clase `UIStyleManager` aplica un tema oscuro al sistema de presentación de la interfaz gráfica. Su propósito es modificar el aspecto visual del usuario final mediante la configuración de colores oscuros, como fondos oscuros y textos brillantes, para mejorar la legibilidad en entornos de baja iluminación o para cumplir con preferencias de diseño modernas. Este método no devuelve un valor (es void), lo que indica que su función es modificar el estado interno del sistema de estilos en lugar de proporcionar un resultado. Es una operación de configuración que afecta a la apariencia global de la interfaz, y suele ser llamada cuando se selecciona un tema oscuro por parte del usuario o por defecto según las preferencias del sistema.
- **loadFont**(String, float): Font — El método `loadFont` de la clase `UIStyleManager` carga y devuelve una instancia de tipo `Font` que se utiliza para definir el estilo visual de texto en la interfaz de usuario. Su propósito principal es obtener una fuente grafica predefinida o personalizada, permitiendo establecer propiedades como tamaño, peso, tipo de letra y otros atributos estéticos. Esta operación es fundamental para mantener la consistencia visual del diseño de la aplicación. Es importante

destacar que el método puede estar diseñado para gestionar la carga eficiente de fuentes (por ejemplo, mediante caching) para evitar repeticiones innecesarias y optimizar el rendimiento en entornos con múltiples pantallas o componentes gráficos.

- **applyRoundedFrame**(JFrame, int, int): void — La méthode `applyRoundedFrame` de la clase `UIStyleManager` aplica un efecto de marco redondeado a componentes de interfaz de usuario. Su propósito es modificar el estilo visual de los elementos gráficos para que tengan bordes redondeados, mejorando así la apariencia y el diseño moderno de la interfaz. Esta operación no devuelve ningún valor (su tipo es `void`), lo que indica que su función se centra en efectos visuales aplicados directamente al elemento afectado. Es un método clave para personalizar estilos de UI, especialmente en entornos donde se requiere una estética más elegante o atractiva mediante bordes redondeados.
- **componentResized**(java.awt.event.ComponentEvent): void
- **qualityImageDraw**(Graphics, Component, Image): void

AddTaskView

Clase AddTaskView que permite agregar tareas mediante una interfaz gráfica y maneja eventos de acción.

extiende `JFrame`

Métodos:

- **AddTaskView**(JPanel): public — La clase AddTaskView representa una vista para agregar tareas en una aplicación. Su propósito principal es proporcionar una interfaz de usuario donde el usuario puede crear y enviar nuevas tareas. El método asociado devuelve un valor público, lo que indica que su resultado (por ejemplo, el objeto de la tarea creada o un estado de éxito/fallo) está accesible desde fuera de la clase. Esta estructura permite integrar la funcionalidad de creación de tareas con el resto del sistema mediante llamadas públicas. Es un componente clave en la capa de presentación responsable de gestionar la entrada del usuario y su conversión en acciones lógicas dentro de la aplicación.

- **AbstractAction**(void): new — La clase AddTaskView.AbstractAction representa una acción abstracta definida dentro de la vista AddTaskView, encargada de proporcionar una estructura básica para manejar operaciones relacionadas con la creación o gestión de tareas. Su propósito es servir como base para acciones específicas (como agregar una tarea), permitiendo reutilizar código común y mantener el diseño modular. Esta clase abstracta no implementa directamente la lógica de negocio, sino que actúa como un template que debe ser extendido por subclases con comportamientos específicos según las necesidades del flujo de trabajo. Es importante destacar que se utiliza en contexto de una vista de formulario (AddTaskView), donde cada acción puede estar vinculada a un evento de usuario (por ejemplo, clic en un botón). La clase permite que el método "returns new" genere una instancia nueva de una acción específica, facilitando la creación dinámica y adaptable de operaciones asociadas al manejo de tareas.
- **actionPerformed**(ActionEvent): void — El método `actionPerformed` de la clase `AddTaskView` se encarga de manejar la acción generada cuando se activa un evento de interfaz de usuario, como un botón de agregar tarea. Su propósito es procesar la entrada del usuario (por ejemplo, al pulsar el botón "Agregar"), validar los datos introducidos y luego enviar o almacenar esa información en el sistema de gestión de tareas. Este método no devuelve ningún valor (es de tipo `void`) y está típicamente asociado a un componente de eventos como un botón o una acción específica del formulario. Es fundamental para la funcionalidad interactiva del formulario de añadir tarea, asegurando que se realice la operación correspondiente al momento de interactuar con el usuario.
- **AddTaskView**(void): public — La clase AddTaskView representa una vista para agregar tareas en una aplicación. Su propósito principal es proporcionar una interfaz de usuario donde el usuario puede crear y enviar nuevas tareas. El método asociado devuelve un valor público, lo que indica que su resultado (por ejemplo, el objeto de la tarea creada o un estado de éxito/fallo) está accesible desde fuera de la clase. Esta estructura permite integrar la funcionalidad de creación de tareas con el resto del sistema mediante llamadas públicas. Es un componente clave en la capa de presentación responsable de gestionar la entrada del usuario y su conversión en acciones lógicas dentro de la aplicación.
- **main**(String[]): void

LoginView

LoginView es una clase que contiene un método principal, un constructor para AbstractAction y un método actionPerformed para manejar eventos de acción.

Métodos:

- **main**(void): void — La clase LoginView contiene un método principal llamado main que no devuelve ningún valor (void). Este método tiene como propósito iniciar la aplicación de autenticación, probablemente cargando una interfaz de inicio de sesión para que el usuario ingrese sus credenciales. Como es un método estático en un objeto de vista (LoginView), se ejecuta directamente al iniciar la aplicación y sirve como punto de entrada principal. Sin embargo, no se proporciona información adicional sobre el flujo de control, las validaciones o la interacción con componentes gráficos, lo que indica que el código podría estar en una etapa inicial o incompleto.
- **AbstractAction**(void): new — La clase `LoginView.AbstractAction` representa una acción abstracta utilizada dentro del componente de vista de autenticación (`LoginView`) para manejar las operaciones relacionadas con el inicio de sesión. Su propósito es definir un patrón de comportamiento común que puede ser heredado por subclases específicas, permitiendo la personalización del flujo de autenticación (por ejemplo, validación de credenciales, redirección tras el éxito o manejo de errores). Esta clase no implementa directamente la lógica de inicio de sesión, sino que proporciona una estructura básica para acciones como "ingresar" o "validar", lo cual facilita la mantenibilidad y el reutilizamiento del código. Es importante destacar que se utiliza en contexto de una vista, sugiriendo un enfoque basado en eventos o acciones dentro de una interfaz gráfica, típico en aplicaciones con arquitectura MVC (Modelo-Vista-Controlador).
- **actionPerformed**(ActionEvent): void — El método `actionPerformed` de la clase `LoginView` es un manejar de eventos que se activa cuando se produce una acción en el componente de vista de inicio de sesión (por ejemplo, al hacer clic en un botón de "Iniciar sesión"). Su propósito principal es procesar la entrada del usuario, como las credenciales de acceso, y coordinar la validación o autenticación correspondiente. Aunque el método no devuelve un valor (es decir, `void`), ejecuta lógica interna que puede incluir la verificación de datos, la comunicación con un servicio

de autenticación o la navegación hacia otra vista si las credenciales son válidas. Es un componente clave en la interfaz de usuario para garantizar que el flujo de inicio de sesión funcione de forma reactiva y segura.

MainView

La clase MainView es un controlador principal que inicializa la interfaz gráfica, configura el estilo de usuario, crea la ventana principal y establece el diseño de la interfaz.

extiende `JFrame`

Métodos:

- **MainView**(LoginController): public — La clase MainView representa una vista principal en una aplicación Java, probablemente utilizada como punto de entrada o interfaz gráfica para el sistema. El método asociado devuelve un valor público, lo que indica que está diseñado para ser accesible desde otros componentes del sistema, posiblemente para inicializar la interfaz principal o proporcionar acceso a la pantalla principal de la aplicación. Su propósito principal es establecer el flujo inicial de ejecución y presentar al usuario la vista principal de la aplicación. Es importante destacar que, aunque no se proporciona código detallado, su estructura sugiere un rol central en la navegación o presentación de la interfaz de usuario.
- **initialize**(void): void — La méthode initialize de la clase MainView se encarga de inicializar los componentes y el estado del interfaz principal. Su propósito es establecer la configuración necesaria para que la vista funcione correctamente al ser cargada, como la creación de elementos gráficos, la asignación de eventos o la carga de datos iniciales. Aunque no devuelve ningún valor (su firma indica void), realiza operaciones críticas de inicialización que aseguran el correcto funcionamiento del sistema visual. Es un paso esencial en el ciclo de vida de la vista principal para garantizar que todos los elementos estén preparados antes de que el usuario comience a interactuar con la aplicación.
- **setupUIStyle**(void): void — La méthode setupUIStyle de la clase MainView configura el estilo de interfaz gráfica (UI) del aplicativo. Su propósito es establecer propiedades visuales como colores, fuentes, bordes y otros aspectos estéticos que influyen en la apariencia general de la vista

principal. Esta función se utiliza durante la inicialización para garantizar que el diseño de la interfaz cumpla con las especificaciones de estilo definidas previamente. Es un método estático que no requiere instancias de la clase, y su implementación puede incluir la configuración de recursos como estilos XML, colores por defecto o propiedades de diseño en tiempo de ejecución. La operación es fundamental para mantener una experiencia de usuario consistente y visualmente atractiva.

- **createMainWindow**(void): void
- **setupLayout**(void): void
- **createMessagePanel**(void): JPanel
- **setCurrentViewLabel**(String): void
- **createTaskPanel**(void): JPanel
- **createTaskScrollPane**(JPanel): JScrollPane
- **getTaskPanel**(void): JPanel
- *(+11 métodos más)*

SplashScreen

Clase SplashScreen que tiene un constructor público y un método para mostrar la pantalla de carga.

extiende `JWindow`

Métodos:

- **SplashScreen**(void): public — La clase SplashScreen representa una pantalla de carga inicial que se muestra al iniciar la aplicación. Su propósito principal es proporcionar un estado visual mientras se realiza el inicio o carga de recursos del sistema. El método público declarado en esta clase permite acceder a la instancia o a los elementos visuales de la pantalla de carga, facilitando su uso desde otras partes del código. Esta clase es fundamental para mejorar la experiencia del usuario al ocultar el tiempo de inicialización y mostrar un indicador visual de progreso. Se utiliza comúnmente en aplicaciones móviles o desktop donde se requiere una presentación inicial antes de mostrar la interfaz principal.

- **showSplash**(int): void — El método `showSplash` de la clase `SplashScreen` muestra una pantalla de carga (splash screen) al iniciar la aplicación. Su propósito es presentar un interfaz visual inicial que indica que el sistema está en proceso de inicio o cargando recursos. Este método no devuelve ningún valor (es decir, tiene tipo `void`) y se utiliza típicamente para mostrar un logo, mensaje de estado o diseño gráfico mientras se realizan tareas de inicialización del sistema. Es una práctica común en aplicaciones móviles y de escritorio para mejorar la experiencia del usuario al proporcionar retroalimentación durante el arranque.