Documentación de API Java

12 Clases | 0 Interfaces

Clases

LoginController

Controlador de autenticación que permite iniciar sesión, registrarse, obtener el nombre de usuario y crear tareas.

Métodos:

- **LoginController**(void): public — Clase: LoginController
- **login**(String, String): boolean — El método `login` de la clase `LoginController` verifica las credenciales de acceso (usuario y contraseña) para determinar si el inicio de sesión es válido. Devuelve un valor booleano: `true` si las credenciales son correctas y `false` en caso contrario. Su propósito principal es gestionar la autenticación del usuario, permitiendo o rechazando el acceso al sistema según el resultado del proceso de validación. Es un componente clave en el flujo de seguridad del aplicativo, asegurando que solo usuarios autorizados puedan acceder a funciones protegidas.
- **register**(String, String, String): boolean — El método register de la clase LoginController permite registrar un nuevo usuario en el sistema. Devuelve un valor booleano que indica si el proceso de registro fue exitoso (true) o falló (false). Este método probablemente verifica la validez de los datos de entrada, como nombre de usuario y contraseña, y asegura que no exista un usuario duplicado antes de almacenar la información en la base de datos. Su propósito principal es facilitar la creación de cuentas de acceso seguras, gestionando errores y validaciones durante el proceso de registro.
- **getUsername**(void): String
- **createTask**(String): void

- **launchMainView**(void): void
- **launchAddTask**(void): void

TasksController

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

Métodos:

- **TasksController**(void): private — La clase TasksController contiene un método que devuelve un valor privado, lo cual indica que está diseñado para gestionar operaciones relacionadas con tareas. Sin embargo, el contexto proporcionado no incluye detalles sobre la implementación específica del método o su funcionalidad real. Dado que el método es privado, solo puede ser accedido dentro de la misma clase, lo que sugiere que su propósito es interno y podría usarse para procesar o manipular datos de tareas, como crear, actualizar o eliminar tareas, sin exponer esta lógica al resto del sistema. Sin una vista más detallada del código, no se puede especificar con precisión su comportamiento funcional, pero su diseño indica un enfoque orientado a la encapsulación y el control interno de operaciones de tareas.
- **getInstance**(void): TasksController — La línea de código `TasksController.getInstance` obtiene una instancia única (singleton) de la clase `TasksController`. Su propósito es garantizar que solo exista una instancia de esta clase en toda la aplicación, lo cual es útil para mantener el estado compartido o gestionar tareas de forma centralizada. Esta técnica permite una mejor gestión de recursos y evita la creación redundante de objetos, optimizando el rendimiento y facilitando el acceso controlado a las funciones relacionadas con el manejo de tareas. Es un patrón común en diseño de software conocido como "Singleton".
- **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. Este método no devuelve ningún valor (indica tipo void), lo que implica que su función es modificar el estado interno de la clase, específicamente al establecer una referencia a la vista

principal. Es un componente clave en la gestión de la interfaz del usuario, ya que asegura que la pantalla o panel principal adecuado sea cargado y activo cuando se inicia la funcionalidad de tareas.

- **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 de entrada principal.

Métodos:

- **main**(String[]): void — El método main de la clase Launch sirve como punto de entrada principal de una aplicación Java. Su propósito es iniciar el programa al ejecutarlo directamente desde la línea de comandos. Aunque el código especificado no incluye implementación concreta, su declaración como método que devuelve void indica que no produce un valor de retorno y se encarga principalmente de inicializar el flujo principal del programa (por ejemplo, creando objetos, cargando recursos o iniciando procesos). Es fundamental para la ejecución de aplicaciones Java, ya que es el primer método que se llama al iniciar la JVM. La existencia de este método en la clase Launch sugiere que esta clase actúa como punto de partida lógico y estructural del sistema.
- **isFirstLaunch**(void): boolean — El código `Launch.isFirstLaunch` es un método perteneciente a la clase `Launch` que devuelve un valor de tipo booleano. Su propósito es determinar si es la primera vez que se ejecuta la aplicación o el proceso de lanzamiento. Si retorna `true`, indica que el sistema ha sido iniciado por primera vez y no ha habido una ejecución

anterior; en caso contrario, devuelve `false` cuando ya se ha realizado al menos una ejecución previa. Este método es útil para implementar funcionalidades como la presentación de una pantalla de bienvenida, la carga inicial de configuraciones o el registro de uso del sistema. Es un mecanismo común en aplicaciones móviles y desktop para optimizar el rendimiento y mejorar la experiencia del usuario mediante comportamientos condicionales según si es la primera vez que se inicia la app.

- **setSplashShown**(void): void — El método `setSplashShown` de la clase `Launch` establece si se muestra o no la pantalla de carga (splash screen) al iniciar la aplicación. Este método recibe un parámetro booleano que indica si el splash debe ser mostrado (`true`) o oculto (`false`). Su propósito principal es permitir al desarrollador controlar la visualización del inicio de sesión o pantalla de carga, lo cual puede ser útil para ajustes de entorno de desarrollo, pruebas o configuraciones específicas donde se desee evitar mostrar la pantalla de carga. Es un método de configuración que no devuelve valor (void), por lo que su efecto es inmediato en el estado visual del lanzador de la aplicación.

AppState

AppState es una clase que contiene un método principal y métodos para determinar si es la primera vez que se inicia la aplicación y para marcar que el pantalla de carga ha sido mostrada.

Métodos:

- **main**(String[]): void — El método main de la clase AppState inicializa y ejecuta la aplicación, sirviendo como punto de entrada principal del programa. Su propósito es establecer el estado inicial de la aplicación y coordinar el flujo de control para que se inicie correctamente. Aunque el código específico no está disponible, en un contexto típico, este método podría crear instancias de componentes clave, cargar configuraciones o iniciar hilos de ejecución. Es fundamental para garantizar que toda la aplicación se inicie en un estado coherente y funcional desde el inicio.
- **isFirstLaunch**(void): boolean — La propiedad `AppState.isFirstLaunch` es un método o campo de la clase `AppState` que devuelve un valor de tipo booleano. Su propósito principal es determinar si una aplicación ha

sido lanzada por primera vez en su vida o si ya ha sido ejecutada anteriormente. Esto permite a la aplicación detectar el estado inicial del usuario y realizar acciones específicas, como mostrar un tutorial, configuraciones iniciales o pantallas de bienvenida, solo durante la primera ejecución. Es útil para optimizar la experiencia del usuario mediante comportamientos personalizados según si es la primera vez que se usa la app. Esta lógica suele ser implementada mediante una variable de estado que se marca como verdadera al primer lanzamiento y se cambia a falso en versiones posteriores, ya sea mediante almacenamiento en preferencias o en un archivo persistente.

- **setSplashShown**(void): void — El método `AppState.setSplashShown` de la clase `AppState` establece un estado que indica si se ha mostrado o no la pantalla de carga (splash screen). Este método recibe un valor booleano como parámetro y actualiza internamente el estado de la aplicación para rastrear si el fragmento inicial de presentación ha sido exhibido. Su propósito principal es mantener un registro del estado visual de inicio de la aplicación, lo cual puede ser útil para evitar mostrar repetidamente la pantalla de carga al iniciar la app o para controlar el flujo de carga en interfaces de usuario móviles o de escritorio. El método no devuelve ningún valor (void), por lo que solo actúa modificando un estado interno. Es una operación fundamental en el manejo del ciclo de vida de una aplicación móvil, asegurando una experiencia de inicio más eficiente y coherente.

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 una referencia numérica única para identificar cada instancia de la clase Task, permitiendo así distinguir entre diferentes tareas en el sistema. Este identificador puede usarse para consultas, actualizaciones o búsquedas eficientes en estructuras de datos como listas o bases de datos.

Es un método accesor que no recibe parámetros y es accesible públicamente.

- **getUserId**(void): int — El método getUserId de la clase Task devuelve un valor de tipo int. Su propósito es obtener el ID único asociado a un usuario que está relacionado con una tarea. Este método permite recuperar la identificación del usuario responsable o asociado a la tarea, lo cual puede ser útil para autenticación, seguimiento de responsabilidades o gestión de permisos en un sistema de tareas. Es un acceso directo al campo de ID de usuario dentro de la entidad Task, y asume que el valor correspondiente está almacenado o calculado previamente en el objeto.
- **getName**(void): String — El método `getName()` pertenece a la clase `Task` y devuelve un valor de tipo `String`. Su propósito es obtener el nombre asociado a una instancia de la tarea. Este método permite acceder al nombre de la tarea de forma directa y estándar, facilitando su uso en interfaces de usuario o en procesos de gestión de tareas. No requiere parámetros ni efectos secundarios, ya que simplemente devuelve el valor almacenado bajo el atributo `name`. Es un método accesador (getter) común en diseño orientado a objetos para proporcionar una interfaz clara y segura para obtener información fundamental sobre la tarea.
- **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 un usuario en una aplicación. El método declarado como público permite que otras clases accedan y utilicen los datos del objeto `User`, facilitando la interacción desde fuera de la clase. Su propósito principal es estructurar la información básica de un usuario, como nombre, correo electrónico o identificador único, y proporcionar acceso controlado a esos atributos mediante métodos públicos. Es fundamental en el diseño de sistemas que requieran autenticación o gestión de perfiles de usuarios.
- **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 proporcionar acceder de forma segura y directa al valor del nombre de usuario almacenado en la instancia actual. Es un método de acceso (getter) estándar, que no modifica el estado del objeto, y se utiliza comúnmente para recuperar datos en consultas o presentaciones donde se requiera el nombre de usuario del usuario autenticado. La firma del método es `String getUsername()`, lo cual indica que devuelve un valor de tipo cadena y no recibe parámetros.
- **getPassword**(void): String — El método `getPassword` de la clase `User` devuelve una cadena de caracteres que representa la contraseña del usuario. Su propósito es permitir obtener el valor de la contraseña almacenada en el objeto `User`, generalmente para validaciones de autenticación o procesos de seguridad. Es importante destacar que, por razones de seguridad, este método no debe exponer directamente la contraseña en la interfaz o en registros; su uso debe estar restringido a contextos seguros y debe combinarse con prácticas como el hashing para proteger los datos sensibles.
- **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 de una base de datos. Su propósito es recuperar un valor de identificación único (nombre de usuario) almacenado en la base de datos, generalmente mediante una consulta SQL o acceso a un conjunto de registros. Este método devuelve un String que representa el nombre de usuario asociado al contexto actual, y es fundamental para autenticación, autorización o procesos de gestión de sesiones. Es importante verificar si el método incluye manejo de excepciones (por ejemplo, errores de conexión o no encontrado) y si requiere parámetros adicionales como un ID de usuario o clave para localizar correctamente el registro.
- **getPassword**(void): String — El método getPassword de la clase DatabaseService devuelve una cadena que representa la contraseña utilizada para autenticarse en una base de datos. Su propósito es proporcionar el valor de la contraseña de forma segura y controlada dentro del sistema, permitiendo su uso en operaciones de conexión a la base de datos. Es importante destacar que, por razones de seguridad, este método debe usarse con precaución y no debería exponer directamente contraseñas sensibles en el código fuente o en logs. La implementación podría incluir mecanismos de encriptación o manejo de credenciales mediante configuraciones externas para evitar la exposición en tiempo de ejecución.
- **connect**(void): Connection — El método `connect` de la clase `DatabaseService` establece una conexión con una base de datos. Su propósito principal es inicializar y retornar un objeto `Connection` que permite realizar operaciones como consultas, inserciones o actualizaciones en la base de datos. Este método generalmente utiliza propiedades de configuración (como URL, usuario y contraseña) para establecer la conexión mediante una API de acceso a bases de datos (por ejemplo, JDBC). Es esencial que se maneje adecuadamente el manejo de excepciones, especialmente en casos de falla al conectar, y que se asegure de que la conexión sea válida antes de devolverla. Además, puede incluir lógica para reutilizar conexiones o gestionar sesiones activas según el entorno.
- **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, permitiendo aplicar temas oscuros, cargar fuentes, dar marcos redondeados a componentes y manejar eventos de tamaño y dibujo de imágenes de calidad.

Métodos:

- **applyDarkTheme**(void): void — La méthode `applyDarkTheme` de la clase `UIStyleManager` aplica un tema oscuro al interfaz gráfica de usuario (UI). Su propósito es modificar el aspecto visual del sistema mediante el establecimiento de estilos como colores oscuros, contraste bajo y fuentes o componentes adaptados a un entorno nocturno. Esta operación no devuelve valor (tipo void), lo que indica que su función es efectuar una modificación en estado directa del estilo de interfaz, sin generar un resultado explícito. Es fundamental para proporcionar opciones de personalización visual al usuario, mejorando la legibilidad y el confort en entornos con poca luz.
- **loadFont**(String, float): Font — El método `loadFont` de la clase `UIStyleManager` carga y devuelve una instancia de objeto `Font` asociada a un estilo de interfaz gráfica. Su propósito principal es permitir que la aplicación acceda a fuentes personalizadas o predeterminadas para aplicar estilos visuales en componentes de usuario, como etiquetas, botones o campos de entrada. Este método generalmente se encarga de cargar una fuente desde un recurso externo (como un archivo .ttf o .otf) o de obtenerla del sistema según el contexto establecido. Es fundamental para mantener la coherencia visual en aplicaciones GUI y garantizar que las fuentes estén disponibles antes de su uso en la interfaz. La carga se realiza de forma eficiente, posiblemente mediante caché para evitar repeticiones innecesarias, lo que mejora el rendimiento general del sistema de estilo gráfico.

- **applyRoundedFrame**(JFrame, int, int): void — La metodología `applyRoundedFrame` de la clase `UIStyleManager` aplica un estilo de marco redondeado a elementos de interfaz de usuario. Su propósito es modificar el aspecto visual de los componentes gráficos para que tengan bordes redondeados, mejorando así la estética y la experiencia de usuario. Esta operación no devuelve ningún valor (tipo void), lo que indica que su función es exclusivamente modificadora del estado visual actual. Es un método clave en el control de estilos, utilizado comúnmente para personalizar la apariencia de ventanas, botones o contenedores gráficos según los requisitos de diseño.
- **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 en una aplicación de tipo interfaz de usuario (UI) encargada de permitir al usuario añadir nuevas tareas. Su método público permite la creación y retorno de una tarea mediante una interacción con el sistema, como por ejemplo la introducción de datos como título, descripción o fecha límite. Este componente actúa como puente entre la capa de presentación y la lógica de negocio, facilitando la entrada de tareas al modelo de datos. La funcionalidad es fundamental en aplicaciones gestoras de tareas para permitir la creación dinámica de nuevas entradas.
- **AbstractAction**(void): new — La clase `AddTaskView.AbstractAction` es una acción abstracta definida dentro del contexto de la vista `AddTaskView`, que se encarga de proporcionar una estructura base para manipular tareas. Su propósito principal es servir como plantilla o componente genérico para acciones relacionadas con el añadido, edición o gestión de tareas en la interfaz gráfica. Aunque no implementa funcionalidades específicas por sí sola, permite que las subclases concretas

definan comportamientos como crear una nueva tarea o actualizarla. La clase está asociada al método `new` que devuelve una instancia de esta acción, indicando su uso para inicializar acciones de creación de tareas en la vista. Es un elemento clave en el diseño de patrones MVC, ya que separa la lógica de la interfaz del flujo de control.

- **actionPerformed**(ActionEvent): void — El método `actionPerformed` de la clase `AddTaskView` se encarga de manejar la acción generada cuando se activa un evento, como el clic en un botón para agregar una tarea. Su propósito principal es responder a eventos de interfaz de usuario (UI), procesar los datos correspondientes (por ejemplo, texto ingresado por el usuario) y, si es necesario, enviarlos al modelo o a otra parte del sistema para que se cree o actualice una nueva tarea en la lista de tareas. Este método es típicamente parte de un patrón de diseño basado en eventos, como el utilizado en Swing (por ejemplo, con `ActionListener`). Aunque no retorna valor (`void`), puede provocar cambios en el estado del sistema al registrar o actualizar una tarea. Es un componente clave para la interactividad del formulario de añadir tareas.
- **AddTaskView**(void): public — La clase AddTaskView representa una vista en una aplicación de tipo interfaz de usuario (UI) encargada de permitir al usuario añadir nuevas tareas. Su método público permite la creación y retorno de una tarea mediante una interacción con el sistema, como por ejemplo la introducción de datos como título, descripción o fecha límite. Este componente actúa como puente entre la capa de presentación y la lógica de negocio, facilitando la entrada de tareas al modelo de datos. La funcionalidad es fundamental en aplicaciones gestoras de tareas para permitir la creación dinámica de nuevas entradas.
- **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 actúa

como punto de entrada para la aplicación y se encarga de inicializar el proceso de autenticación del usuario. Su propósito es iniciar la interfaz de inicio de sesión, probablemente creando una instancia de la vista LoginView para mostrar el formulario de ingreso de credenciales (usuario y contraseña). Aunque no devuelve un valor, su ejecución inicia la secuencia del sistema de autenticación, lo que permite al usuario ingresar sus datos. Es importante notar que, dado que el método está dentro de una clase de vista, se asume que se integra con componentes gráficos o de presentación, y podría estar diseñado para ejecutarse en un entorno de aplicaciones desktop o móviles basados en Java Swing o JavaFX.

- **AbstractAction**(void): new — La clase `LoginView.AbstractAction` representa una acción abstracta definida dentro del contexto de la vista de autenticación (`LoginView`). Su propósito es proporcionar una estructura base para acciones relacionadas con el proceso de inicio de sesión, como validar credenciales o navegar al siguiente estado después de un login exitoso. Esta clase no implementa funcionalidad específica en sí misma, sino que sirve como plantilla para subclases que deben definir el comportamiento concreto (por ejemplo, cómo se verifica la contraseña o qué vista se muestra tras el éxito del login). Es importante destacar que se utiliza en conjunto con un método de retorno que crea una nueva instancia de esta acción, lo que indica su uso como patrón de diseño para gestionar eventos de interfaz de usuario relacionados con el inicio de sesión.
- **actionPerformed**(ActionEvent): void — La metodología `actionPerformed` del objeto `LoginView` se encarga de manejar la acción generada cuando el usuario interactúa con un componente de interfaz (como un botón de login). Este método es típicamente parte de un controlador de eventos en una aplicación basada en Swing o JavaFX, y su función principal es responder a eventos de entrada del usuario. Al ser invocado por eventos como el clic en un botón de acceso, ejecuta la lógica necesaria para validar las credenciales (por ejemplo, verificar nombre de usuario y contraseña), y luego puede redirigir al usuario a una pantalla principal o mostrar un mensaje de error si las credenciales son incorrectas. Como el método no devuelve ningún valor (`void`), su responsabilidad está en realizar acciones en tiempo real sin afectar el estado de la aplicación mediante un retorno. Es clave para garantizar la interactividad y funcionalidad del sistema de autenticación.

MainView

La clase MainView es una vista principal que se inicializa con un controlador de login y configura el estilo de interfaz, el diseño de la ventana principal y la estructura de la interfaz gráfica.

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 principal para la interacción del usuario. El método asociado devuelve un valor público, lo que indica que está diseñado para ser accesible desde otras clases fuera de su propio paquete, permitiendo acceder a la instancia o estructura principal de la vista. Su propósito es proporcionar una interfaz central para el control y la presentación de la funcionalidad del sistema. No se especifica el tipo de retorno, pero dado que se menciona como método público en un contexto de vista, podría estar relacionado con la creación o acceso a una instancia principal (por ejemplo, un objeto de tipo JFrame o Activity). Es importante destacar que, sin más código, su funcionamiento exacto depende del contexto específico de la aplicación (como el uso de Java Swing, Android o un framework gráfico personalizado).
- **initialize**(void): void — La metodología initialize de la clase MainView se encarga de inicializar los componentes principales de la vista principal. Su propósito es configurar y preparar el entorno gráfico o funcional necesarios para que la interfaz de usuario funcione correctamente al inicio. Esta función no devuelve un valor (su tipo es void), lo que indica que su objetivo es realizar acciones de configuración en lugar de devolver un resultado. Es una operación crucial para garantizar que todos los elementos visuales y lógicos estén listos antes de que el usuario comience a interactuar con la aplicación.
- **setupUIStyle**(void): void — La metodología setupUIStyle de la clase MainView configura el estilo gráfico de la interfaz de usuario principal. Su propósito es establecer propiedades visuales como colores, fuentes, bordes o diseño general que definen la apariencia del componente de vista principal. Esta función es fundamental para garantizar una consistencia

estética y funcional en la experiencia del usuario. Como devuelve void, no retorna un valor, lo que indica que su función está orientada únicamente a modificar el estado visual en lugar de proporcionar datos o resultados. Es un método típico de inicialización de UI en aplicaciones basadas en Java Swing o JavaFX.

- **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 una aplicación. Su propósito es proporcionar un feedback visual durante el tiempo de inicio, mejorando la experiencia del usuario mediante una presentación gráfica mientras se cargan los recursos o se inicia el sistema principal. El método público de esta clase probablemente devuelve una instancia de la pantalla de carga, permitiendo su uso en la secuencia de inicialización de la aplicación. Esta implementación es clave para mantener la percepción de rendimiento y estabilidad del software desde el primer momento de ejecución.
- **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 iniciando o cargando recursos, mejorando así la experiencia del usuario mediante una retroalimentación visual. Este método no devuelve ningún valor (su tipo es `void`), lo que implica que su función es exclusivamente para mostrar información en pantalla y no para devolver un resultado operativo. Es un componente fundamental en la interfaz de inicio de aplicaciones móviles o desktop, ya que proporciona una sensación de respuesta inmediata al arrancar el sistema.