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 una autenticación exitosa se ha realizado. Devuelve un valor booleano: `true` si las credenciales son válidas, y `false` en caso contrario. Su propósito principal es gestionar el proceso de inicio de sesión del sistema, validando el usuario frente a una base de datos o servicio de autenticación externo. Es fundamental para controlar el acceso a áreas protegidas de la aplicación. La implementación puede incluir mecanismos de seguridad como hashing de contraseñas y protección contra intentos múltiples fallidos.
- **register**(String, String, String): boolean — El método register de la clase LoginController se encarga de 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 típicamente valida los datos del usuario (como nombre de usuario, contraseña y correo electrónico), verifica la unicidad de estos campos para evitar duplicados, y, si todo está correcto, guarda la información en la base de datos. Es un componente clave del flujo de autenticación, permitiendo a los usuarios acceder al sistema posteriormente. La implementación puede incluir validaciones de entrada,

manejo de excepciones y control de errores para garantizar la integridad y seguridad del registro.

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

TasksController

Controlador de tareas que permite obtener y gestionar una lista de tareas desde una base de datos mediante métodos como cargar tareas o obtenerlas por criterios.

Métodos:

- **TasksController**(void): private — La clase TasksController contiene un método que devuelve un valor privado. Sin embargo, el código proporcionado no incluye implementación específica ni detalles sobre la lógica o funcionalidad del controlador. Por lo tanto, no se puede determinar con precisión su propósito técnico o comportamiento en ejecución. Se requiere más información para analizar detalladamente su función y estructura.
- **getInstance**(void): TasksController — La línea de código `TasksController.getInstance` obtiene una instancia única (singleton) de la clase `TasksController`. Su propósito es proporcionar acceso global a un controlador que gestiona las tareas, garantizando que solo exista una copia de esta clase en toda la aplicación. Esto es útil para mantener el estado compartido entre diferentes componentes del sistema y evitar la creación múltiple de instancias, lo cual optimiza el uso de recursos y mantiene la coherencia del estado de las tareas. Es un patrón común (singleton) utilizado en diseño de software para facilitar el acceso a un objeto centralizado.
- **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 se mostrará al usuario dentro del controlador de tareas. Este método no devuelve un

valor, por lo que su firma es void. Es fundamental para el flujo de navegación y renderizado de la aplicación, ya que define qué componente visual será el principal en la presentación de las tareas. La implementación específica puede depender de la arquitectura de interfaz (por ejemplo, uso de un framework como JavaFX o Swing) y del diseño de componentes dentro del controlador.

- **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, configurar la muestra inicial 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 en Java. Su propósito es iniciar la ejecución del programa, normalmente mediante la inicialización de objetos, el establecimiento de la configuración inicial o el lanzamiento de operaciones clave. Como retorna void, no devuelve un valor específico al sistema; su función es simplemente iniciar el flujo de ejecución. Es importante destacar que este método debe estar presente en una clase pública para que Java pueda identificarla como punto de entrada, y generalmente se define con el nombre "main" y un parámetro de tipo String[] para recibir argumentos de línea de comandos.
- **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. Devuelve `true` cuando el sistema detecta que no ha habido una ejecución anterior, y `false` en caso contrario. Este método es útil para implementar funcionalidades como pantallas de bienvenida, configuraciones iniciales o instrucciones para el primer uso del software, permitiendo personalizar la experiencia del usuario según si es su primera sesión o no.

- **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 no devuelve ningún valor. Este método permite controlar visiblemente el comportamiento inicial de la interfaz gráfica, determinando si se muestra una animación o mensaje de carga antes de que se inicie el proceso principal. Es útil para personalizar la experiencia de inicio según las necesidades del sistema o del usuario final.

AppState

Clase AppState que contiene métodos para determinar si es la primera ejecución, mostrar la pantalla de carga y el punto de entrada principal.

Métodos:

- **main**(String[]): void — El método main de la clase AppState inicializa y ejecuta el estado principal de la aplicación. Su propósito es servir como punto de entrada principal del programa, donde se configuran las condiciones iniciales, se cargan los recursos necesarios y se inicia el ciclo principal de operación de la aplicación. Este método no devuelve ningún valor (es void), lo que indica que su función es ejecutar acciones en lugar de devolver un resultado. Es fundamental para el arranque correcto del sistema y actúa como el núcleo inicial del flujo de control de la aplicación.
- **isFirstLaunch**(void): boolean — La propiedad `AppState.isFirstLaunch` es un método o propiedad de la clase `AppState` que devuelve un valor booleano. Su propósito es indicar si la aplicación ha sido lanzada por primera vez en una sesión específica. Si el valor es `true`, significa que el usuario está ejecutando la aplicación por primera vez; si es `false`, indica que ya se ha iniciado anteriormente. Este mecanismo es útil para implementar funcionalidades como pantallas de bienvenida, configuraciones iniciales o tutoriales que deben mostrarse solo una vez. Es

importante tener en cuenta que el estado puede depender de la lógica interna de `AppState`, por ejemplo, mediante almacenamiento persistente (como SharedPreferences o un archivo) para mantener la información entre lanzamientos.

- **setSplashShown**(void): void — La llamada `AppState.setSplashShown` establece un estado en la clase `AppState` que indica si se ha mostrado o no la pantalla de carga (splash screen). Este método tiene como propósito marcar el indicador de que el fragmento inicial de la aplicación ya fue exhibido al usuario, lo cual puede ser útil para evitar mostrar repetidamente la pantalla de carga al iniciar la app. El método no devuelve un valor, por lo que su firma es `void`. Es una operación de estado que se utiliza comúnmente en aplicaciones móviles o desktop para optimizar el flujo inicial y mejorar la experiencia de usuario mediante la gestión eficiente del estado visual.

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 identificar y distinguir entre diferentes tareas en el sistema. Es un método accesible público, comúnmente utilizado para consultas o operaciones que requieren referenciar una tarea por su clave única. No modifica el estado del objeto, ya que solo accede a un atributo de datos existente.
- **getUserId**(void): int — El método getUserId de la clase Task devuelve un valor de tipo int. Su propósito es obtener un identificador único asociado al usuario que está relacionado con una tarea. Este identificador puede utilizarse para realizar consultas o operaciones específicas basadas en el usuario propietario de la tarea. Es importante destacar que este método asume que cada instancia de Task está vinculada a un usuario específico, y que el valor devuelto representa el ID del usuario asociado. No se

especifica si el identificador proviene de una base de datos, una configuración interna o es generado dinámicamente.

- **getName**(void): String — El método getName de la clase Task devuelve una cadena de caracteres que representa el nombre de la tarea. Su propósito es proporcionar un valor textual único o descriptivo que identifica la tarea dentro del sistema. Este método permite acceder al nombre de la instancia actual de la clase Task, lo cual puede ser útil para la visualización, el logging o la gestión de tareas en interfaces de usuario o sistemas de control. No recibe parámetros y es un método de acceso público, simple y directo, cuya implementación específica depende del diseño interno de la clase Task.
- **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 de usuario con métodos públicos que permiten acceder o manipular propiedades relacionadas con el usuario. Su propósito principal es representar la entidad de un usuario en una aplicación, proporcionando una interfaz clara y accesible para operaciones como obtener datos del usuario. Dado que los métodos son públicos, cualquier componente de la aplicación puede interactuar directamente con esta clase para acceder a sus atributos o realizar acciones sobre el objeto User. Esta estructura es fundamental en

sistemas que requieren gestionar usuarios, garantizando una jerarquía clara y un acceso controlado a la información del usuario.

- **getUsername**(void): String — El método getUsername de la clase User obtiene y devuelve el nombre de usuario como un objeto de tipo String. Su propósito es proporcionar acceder al identificador único del usuario almacenado dentro del objeto User. Es un método accesador (getter) que permite recuperar el valor del atributo username sin modificarlo, y se utiliza comúnmente para autenticación, autorización o presentación de datos del usuario en la interfaz. El método no recibe parámetros y siempre devuelve una cadena de caracteres representando el 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 permitir el acceso al valor almacenado de la contraseña, generalmente para autenticación o verificación de credenciales. Es importante destacar que, por razones de seguridad, este método no debe exponer directamente la contraseña en el código fuente ni ser utilizado en entornos donde se pueda acceder a datos sensibles sin cifrado adecuado. La implementación debe seguir buenas prácticas de seguridad, como evitar el almacenamiento claro de contraseñas y utilizar hashing o autenticación basada en tokens.
- **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 recuperar el nombre de usuario desde una base de datos. Su propósito es obtener un valor de cadena (String) que representa el nombre de usuario asociado a una entidad específica, generalmente mediante una consulta SQL o acceso a un repositorio de

datos. Este método es fundamental para autenticación o identificación de usuarios en aplicaciones basadas en bases de datos. Se asume que requiere una conexión activa a la base de datos y puede depender de parámetros como el ID del usuario, para garantizar la precisión del resultado. Es un componente clave en el flujo de autenticación y gestión de sesiones.

- **getPassword**(void): String — El método getPassword de la clase DatabaseService devuelve una cadena de caracteres que representa la contraseña utilizada para autenticarse en una base de datos. Su propósito es proporcionar el valor de la contraseña necesaria para establecer una conexión segura con la base de datos, generalmente para fines de configuración o autenticación. Es importante destacar que este método debería usarse con precaución debido al riesgo de exposición de credenciales sensibles; por ello, en entornos reales se recomienda almacenar las contraseñas de forma cifrada y utilizar mecanismos de manejo seguro de secretos (como configuraciones externas o sistemas de gestión de claves).
- **connect**(void): Connection — El método `connect` de la clase `DatabaseService` establece una conexión con una base de datos. Su propósito principal es inicializar y devolver 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, contraseña) para autenticarse y establecer una conexión a través de un driver JDBC. Es fundamental que el método maneje adecuadamente excepciones de conexión y garantice que la conexión sea válida antes de su devolución. Además, en entornos productivos, podría incluirme mecanismos de reintentar la conexión o de verificar el estado del servicio de base de datos.
- **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 redimensionamiento y dibujo de imágenes con calidad.

Métodos:

- **applyDarkTheme**(void): void — La metodología `applyDarkTheme` de la clase `UIStyleManager` aplica un tema oscuro al interfaz gráfica de usuario (UI). Su propósito es modificar el aspecto visual del sistema de estilo mediante la configuración de colores oscuros, como fondos oscuros y textos más luminosos, para mejorar la visibilidad en condiciones de baja iluminación o en pantallas con alta contraste. Este método no devuelve un valor (es void), lo que indica que su función es modificar el estado del UI directamente sin producir un resultado explícito. Es una acción de configuración que puede ser llamada durante la inicialización del sistema para establecer el tema por defecto o en respuesta a preferencias del usuario.
- **loadFont**(String, float): Font — El método `loadFont` de la clase `UIStyleManager` carga y devuelve una instancia de tipo `Font`. Su propósito principal es obtener una fuente tipográfica configurada para ser utilizada en la presentación gráfica de interfaces de usuario. Este método generalmente se encarga de cargar una fuente desde un recurso externo (como un archivo de fuentes o una definición interna) y aplicarla al sistema de estilo de la interfaz. Es fundamental para garantizar que los componentes visuales utilicen fuentes consistentes, compatibles y personalizadas según las especificaciones del diseño UI. La carga se realiza de forma eficiente mediante un manejo adecuado de recursos, evitando el uso duplicado o la sobrecarga en aplicaciones con múltiples pantallas o componentes.
- **applyRoundedFrame**(JFrame, int, int): void — La méthode `applyRoundedFrame` del objeto `UIStyleManager` aplica un efecto de marco redondeado a los componentes de la interfaz gráfica. Su propósito es modificar el estilo visual de elementos UI para que tengan bordes

redondeados, mejorando la apariencia moderna y estética. Esta operación afecta directamente las propiedades visuales del componente, como el radio de los bordes (cornerRadius), y se utiliza comúnmente en diseños con interfaces responsivas o temáticas. La método no devuelve un valor (void), lo que indica que su efecto se aplica inmediatamente al componente sin generar un resultado devuelto. Es una acción de configuración visual que depende del contexto específico del sistema de estilo UI.

- **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 dedicada a la creación de tareas y proporciona un método público que permite agregar nuevas tareas al sistema. Su propósito principal es ofrecer una interfaz para permitir al usuario introducir detalles de una tarea (como título, descripción o fecha) y luego procesar esta información mediante el método público correspondiente. Esta clase está diseñada para integrarse en una aplicación de gestión de tareas, facilitando la interacción entre el frontend (interfaz de usuario) y el backend (lógica de negocio). No se especifica el tipo exacto de dato devuelto por el método, pero su naturaleza pública sugiere que puede ser accesible desde otros componentes del sistema.
- **AbstractAction**(void): new — La clase AddTaskView.AbstractAction representa una acción abstracta utilizada dentro de la vista AddTaskView para manejar operaciones relacionadas con la creación o gestión de tareas. Su propósito principal es proporcionar un punto de entrada común para diferentes acciones (como agregar, editar o cancelar una tarea), permitiendo que las subclases específicas implementen el comportamiento detallado según el caso de uso. Esta estructura promueve la reutilización del código y facilita la mantenibilidad al separar la lógica de la interfaz

gráfica. Es importante destacar que esta clase no define su propio comportamiento, sino que actúa como una plantilla para acciones concretas implementadas en otras clases hijas, lo cual es clave para el diseño orientado a objetos y la modularidad del sistema.

- **actionPerformed**(ActionEvent): void — El método `actionPerformed` de la clase `AddTaskView` se encarga de manejar la acción generada cuando se activa un componente de interfaz de usuario (por ejemplo, un botón) asociado a la vista de agregado de tareas. Su propósito principal es responder a eventos del usuario, como el clic en un botón para añadir una nueva tarea, y coordinar la lógica necesaria para procesar dicha acción. Este método generalmente invoca métodos del modelo o controlador para crear o registrar una nueva tarea basada en los datos ingresados por el usuario en la vista. Es un componente clave de la arquitectura MVC, ya que actúa como intermediario entre la interfaz y la lógica de negocio. Aunque no tiene retorno (es void), es responsable de ejecutar operaciones específicas dependiendo del evento recibido.
- **AddTaskView**(void): public — La clase AddTaskView representa una vista dedicada a la creación de tareas y proporciona un método público que permite agregar nuevas tareas al sistema. Su propósito principal es ofrecer una interfaz para permitir al usuario introducir detalles de una tarea (como título, descripción o fecha) y luego procesar esta información mediante el método público correspondiente. Esta clase está diseñada para integrarse en una aplicación de gestión de tareas, facilitando la interacción entre el frontend (interfaz de usuario) y el backend (lógica de negocio). No se especifica el tipo exacto de dato devuelto por el método, pero su naturaleza pública sugiere que puede ser accesible desde otros componentes del sistema.
- **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 se encarga de iniciar la aplicación de autenticación, típicamente mediante la creación de una interfaz de inicio de sesión para que el usuario ingrese sus credenciales. Su propósito es actuar como punto de entrada principal del sistema, cargando y ejecutando las funcionalidades necesarias para el acceso al sistema. Es importante destacar que, dado que el método main está dentro de una clase denominada LoginView, sugiere que la vista de inicio de sesión se encarga directamente de gestionar el flujo de autenticación del usuario. Sin embargo, como no se proporciona código adicional, su comportamiento exacto depende de la lógica implementada en dicho método.
- **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 principal es definir un comportamiento común que pueden heredar otras clases específicas, permitiendo estandarizar la lógica de procesamiento durante el login (por ejemplo, validación de credenciales, gestión de sesiones o redirección). Como clase abstracta, no puede instanciarse directamente; su uso depende de subclases que implementen la lógica concreta. Esta estructura facilita la mantenibilidad y extensibilidad del código al separar la definición general de una acción (como el inicio de sesión) de su implementación específica, lo cual es clave en arquitecturas basadas en vistas o patrones MVC.
- **actionPerformed**(ActionEvent): void — La método `actionPerformed` de la clase `LoginView` es un manejador 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 entrada). Este método no devuelve ningún valor (es void) y está diseñado para procesar la solicitud de autenticación, como validar las credenciales del usuario (usuario y contraseña), verificar si los datos son correctos según las reglas definidas, y, en función del resultado, dirigir a la aplicación hacia el estado adecuado (por ejemplo, mostrar una pantalla principal o mostrar un mensaje de error). Es fundamental para la interacción entre la interfaz gráfica y el sistema de autenticación, actuando como punto de entrada para el procesamiento de eventos relacionados con el inicio de sesión.

MainView

La clase MainView es un controlador principal que inicializa la interfaz de usuario, configura el estilo gráfico, establece el diseño y crea la ventana principal mediante métodos como initialize, setupUIStyle, createMainWindow y setupLayout.

extiende `JFrame`

Métodos:

- **MainView**(LoginController): public — La clase MainView contiene un método público que probablemente sirve como punto de entrada principal de una aplicación, encargado de mostrar la vista principal o el interfaz gráfica de usuario (GUI) inicial. Su propósito es iniciar la ejecución de la aplicación y presentar la pantalla principal al usuario. Dado que se especifica que el método es público, puede ser accesible desde otras clases, lo cual permite una integración más fluida con otros componentes del sistema. No se proporciona código detallado, por lo que su funcionalidad específica depende del contexto de la aplicación; sin embargo, en un entorno típico, esta clase gestiona el flujo inicial de la interfaz y puede encargarse de cargar recursos o inicializar componentes esenciales.
- **initialize**(void): void — La méthode initialize de la clase MainView se encarga de inicializar el estado y los componentes principales de la vista principal. Su propósito es establecer la configuración inicial del interfaz gráfico de usuario (GUI), como la creación o asignación de elementos visuales, la carga de datos iniciales o la configuración de eventos. Esta metodología es común en aplicaciones basadas en JavaFX, donde el método initialize se ejecuta al inicio para asegurar que todos los elementos de la interfaz estén correctamente preparados antes de que el usuario interactúe con la vista. La firma del método devuelve void, indicando que no produce un valor y que su función es exclusivamente de inicialización.
- **setupUIStyle**(void): void — La metodología `setupUIStyle` de la clase `MainView` configura el estilo gráfico de la interfaz de usuario. Su propósito es establecer propiedades visuales como colores, fuentes, bordes y otros aspectos estéticos que determinan la apariencia general del diseño principal. Esta función no retorna un valor (su tipo es `void`) y se ejecuta

durante la inicialización para asegurar una presentación coherente y visualmente atractiva de la interfaz. Es fundamental para mantener la consistencia de diseño en aplicaciones basadas en Java Swing o JavaFX, donde el estilo UI puede influir directamente en la experiencia del usuario.

- **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 principal es presentar un diseño visual durante el tiempo necesario para cargar recursos o realizar operaciones de inicio, mejorando así la experiencia del usuario al ocultar el proceso de arranque. El método asociado a esta clase devuelve valor público, lo que indica que puede ser accedido desde otras clases fuera de su propio ámbito, permitiendo integrarlo en el flujo principal de la aplicación. Es importante destacar que esta clase no realiza operaciones lógicas complejas ni maneja estados del sistema; su función es exclusivamente visual y temporal, sirviendo como una capa inicial para mantener la interfaz amigable mientras se inicia el resto del sistema.
- **showSplash**(int): void — El método `showSplash` de la clase `SplashScreen` muestra una pantalla de carga (splash screen) al inicio de

la aplicación. Su propósito es presentar un interfaz visual inicial que indica que el sistema está iniciando o cargando recursos. Este método no devuelve ningún valor (es decir, tiene tipo void), lo que implica que su función es únicamente mostrar una vista y no procesar ni retornar datos. Es un componente clave en la experiencia de usuario para mejorar la percepción de rendimiento al ocultar el tiempo de inicio.