1. Paquete Principal (Unnamed Package)

- **1.1 CrearTarea**
  - Clase que representa la ventana para crear una nueva tarea.
- **1.2 CrearTareaPropia**
  - Clase que permite la creación de tareas propias para un usuario en la interfaz gráfica de Taskflow.
- **1.3 CrearTareaUsuario**
  - Clase que representa la interfaz de usuario para la creación de tareas en el sistema.
- **1.4 DetallesTarea**
  - Ventana que muestra los detalles de una tarea específica.
- **1.5 DetallesTareaPropia**
  - Ventana que muestra los detalles de una tarea específica.
- **1.6 EditarNivelUsuario**
  - Clase que permite editar el nivel de un usuario específico en la aplicación.
- **1.7 EditarPerfil**
  - Ventana para editar el perfil del usuario actual.
- **1.8 EditarPerfilUsuario**
  - Clase que representa la ventana para editar el perfil del usuario.
- **1.9 EditarTarea**
  - Ventana para editar los detalles de una tarea específica**.**
- **1.10 EditarTareaCompletada**
  - Ventana para editar las tareas completadas de un usuario.
- **1.11 EditarTareaCompletadaUsuario**
  - Ventana para editar una tarea completada asignada a un usuario.
- **1.12 EditarTareaPropia**
  - Ventana de la interfaz gráfica para editar tareas propias.
- **1.13 EditarTareaPropiaUsuario**
  - Ventana de la interfaz gráfica para editar tareas de un usuario.
- **1.14 EditarTareaUsuario**
  - Ventana de interfaz gráfica para editar tareas de un usuario en la aplicación.
- **1.15 Escalar**
  - La clase Escalar proporciona métodos para redimensionar imágenes y ajustarlas a un componente gráfico.
- **1.16 Historial**
  - Ventana que muestra el historial de tareas de un usuario.
- **1.17 HistorialUsuario**
  - Ventana que muestra el historial de tareas de un usuario.
- **1.18 Inicio**
  - Clase que representa la ventana principal de la aplicación.
- **1.19 InicioUsuario**
  - Clase que representa la ventana principal de la interfaz de usuario.
- **1.20 Login**
  - Clase que representa la ventana de inicio de sesión en la aplicación.
- **1.21 MisTareas**
  - Clase que representa la ventana de "Mis Tareas", donde se gestionan y visualizan las tareas asignadas al usuario.
- **1.22 MisTareasUsuario**
  - Clase que representa la ventana principal de la interfaz de usuario para las tareas de un usuario.
- **1.23 PanelRound**
  - Clase que permite crear paneles con esquinas redondeadas personalizables.

- **1.24 Perfil**
  - Clase que muestra el perfil de usuario en la interfaz gráfica.
- **1.25 PerfilUsuario**
  - Clase que representa la ventana del perfil de un usuario en la interfaz gráfica.
- **1.26 PerfilView**
  - Clase que representa la vista del perfil de un usuario en la interfaz gráfica.
- **1.27 Registro**
  - Clase que representa la ventana de registro de un usuario en la interfaz gráfica.
- **1.28 Tarea**
  - Clase que representa una tarea en un sistema de gestión de tareas.
- **1.29 TareaRenderer**
  - Clase para personalizar la representación de celdas en una tabla.
- **1.30 TareaRenderer2**
  - Clase para personalizar la representación de celdas en una tabla.
- **1.31 TareasActuales**
  - Ventana que muestra las tareas actuales de un usuario.
- **1.32 TareasActualesUsuario**
  - Ventana para gestionar las tareas actuales de un usuario.
- **1.33 Usuario**
  - Representa a un usuario en el sistema, con su información básica y métodos para gestionar su registro, edición y eliminación en archivos.
- **1.34 Usuarios**
  - Clase que representa la ventana principal para gestionar y visualizar los usuarios.

---

2. Paquete `raven_cell`

- **2.1 ActionButton**
  - Clase personalizada que extiende `JButton` para crear un botón circular con un cambio visual de color al presionar el mouse.
- **2.2 PanelAction**
  - Panel personalizado que contiene botones de acción (Editar, Eliminar, Ver) para ser utilizados en una tabla.
- **2.3 TableActionCellEditor**
  - Clase personalizada que extiende `DefaultCellEditor` para permitir la edición de celdas en una tabla con botones de acción específicos.
- **2.4 TableActionCellRender**
  - Clase personalizada que extiende `DefaultTableCellRenderer` para renderizar celdas en una tabla con un panel que contiene botones de acción.
- **2.5 TableActionEvent**
  - Interfaz para manejar eventos de acción en una tabla, como editar, eliminar y ver.
- **2.6 test**
  - `JFrame` de prueba para `raven_cell`.

---

3. Paquete `raven_cell2`

- **3.1 ActionButton**
  - Clase que extiende `JButton` y personaliza su apariencia visual para simular un botón circular.
- **3.2 PanelAction**
  - Clase que representa un panel de acción dentro de la interfaz de usuario.
- **3.3 TableActionCellEditor**
  - Clase que proporciona un editor personalizado para las celdas de una tabla.
- **3.4 TableActionCellRender**

- Clase que extiende `DefaultTableCellRenderer` y proporciona un renderizador personalizado para las celdas de acción de la tabla.
- **3.5 TableActionEvent**
- Interfaz que define los eventos de acción para las celdas de una tabla.
- **3.6 test2**
- `JFrame` de prueba para `raven_cell2`.

1. Paquete principal (Unnamed Package):

Este paquete contiene las clases principales que inician el proyecto.

1.1.Clase CrearTarea

Descripción:

La clase **`CrearTarea`** extiende de `JFrame` y representa la ventana principal de la interfaz de usuario para la creación de nuevas tareas. Su objetivo es proporcionar una interfaz gráfica donde los usuarios puedan ingresar información sobre una nueva tarea.

Campos:

- **`usuario`**: Objeto de tipo `Usuario` que representa al usuario actual.
- **`userHome`**: Cadena que almacena la ruta del directorio del usuario en el sistema operativo.
- **`lista`**: Lista de usuarios cargados desde un archivo llamado `fileUsuarios`.

Constructor:

- **`CrearTarea()`**: Constructor que inicializa la ventana, escala las imágenes para ajustarlas al tamaño de los componentes, obtiene el usuario actual y llena el ComboBox con los nombres de los usuarios disponibles.

Métodos:

- **`main(String[] args)`**: Método principal que sirve como punto de entrada para la ventana.

Herencia:

La clase hereda de **`JFrame`**, lo que le permite ser una ventana en la interfaz gráfica de Java. También hereda propiedades y métodos de clases más generales como **`Window`** y **`Component`**, lo que le otorga capacidades adicionales, como la manipulación de eventos y el control de su apariencia.

1.2.Clase CrearTareaPropia

Descripción:

La clase **`CrearTareaPropia`** extiende de **`JFrame`** y proporciona una ventana gráfica en la interfaz de usuario de **Taskflow** que permite a los usuarios crear tareas propias. Es parte de la interfaz donde los usuarios pueden interactuar con el sistema para gestionar sus tareas personales.

Campos:

- **`usuario`**: Objeto de tipo `Usuario` que representa al usuario actual dentro del sistema.
- **`userHome`**: Ruta predeterminada del directorio del usuario en el sistema operativo.

Constructor:

- **`CrearTareaPropia()`**: Constructor que inicializa los componentes gráficos de la interfaz, configura las rutas de las tareas y la información del usuario. También ajusta las imágenes para que se escalen correctamente en la interfaz y muestra el nombre del usuario actual en la ventana.

Métodos:

- **`main(String[] args)`**: Método principal que actúa como el punto de entrada para la ventana. Este método permite que la ventana se inicialice y se muestre.

Herencia:

La clase hereda de **`JFrame`**, lo que le permite funcionar como una ventana en la interfaz gráfica. También hereda métodos de las clases **`Window`** y **`Component`**, lo que le otorga la capacidad de manejar eventos y personalizar su apariencia.

1.3.Clase CrearTareaUsuario

Descripción:

La clase **`CrearTareaUsuario`** extiende **`JFrame`** y proporciona una ventana gráfica que permite al usuario crear tareas en el sistema. Es una interfaz en la que el usuario puede interactuar para agregar y gestionar sus tareas dentro de la aplicación.

Campos:

- **`usuario`**: Instancia de la clase **`Usuario`** que contiene los datos inicializados del usuario actual.
- **`userHome`**: Ruta que apunta al directorio base del usuario, generalmente al escritorio del sistema operativo.

Constructor:

- **`CrearTareaUsuario()`**: Constructor que inicializa la interfaz gráfica del usuario. Este constructor también se encarga de escalar las imágenes de los componentes de la ventana y de configurar las rutas necesarias para el almacenamiento de las tareas que el usuario cree.

Métodos:

- **`main(String[] args)`**: Método principal que sirve como punto de entrada para la ejecución de la ventana. Inicializa y muestra la interfaz gráfica.

Herencia:

La clase **`CrearTareaUsuario`** hereda de **`JFrame`**, lo que le permite comportarse como una ventana dentro de la aplicación. También hereda métodos y propiedades de las clases superiores **`Window`** y **`Component`**, lo que le proporciona capacidades para manejar eventos, realizar dibujos en la pantalla y manipular la interfaz de usuario.

1.4.Clase DetallesTarea

Descripción:

La clase **`DetallesTarea`** extiende **`JFrame`** y muestra una ventana que presenta los detalles de una tarea específica. Esta ventana muestra información como el nombre de la tarea, descripción, estado, prioridad, usuario asignado y fecha de creación o vencimiento de la tarea.

Campos:

- **`actual`**: Un objeto de tipo **`Tarea`** que representa la tarea actualmente seleccionada o en ejecución. Es el objeto cuya información será mostrada en la interfaz.

Constructor:

- **`DetallesTarea()`**: Constructor que inicializa los componentes de la ventana, ajusta las imágenes y configura la interfaz para mostrar los detalles de la tarea.

Métodos:

- **`main(String[] args)`**: Método principal que sirve como punto de entrada para la ejecución de la ventana. Inicializa y muestra la interfaz gráfica para los detalles de una tarea.
- **`setTarea(Tarea aux)`**: Establece la tarea que se desea mostrar en la ventana. El parámetro **`aux`** es un objeto de tipo **`Tarea`** que contiene la información de la tarea seleccionada.

Herencia:

La clase **`DetallesTarea`** hereda de **`JFrame`**, lo que le permite comportarse como una ventana dentro de la aplicación. También hereda métodos y propiedades de las clases superiores **`Window`** y **`Component`**, permitiéndole manejar eventos y manipular su interfaz de usuario.

1.5.Clase DetallesTareaPropia

Descripción:

La clase **`DetallesTareaPropia`** extiende **`JFrame`** y se utiliza para mostrar una ventana que presenta los detalles de una tarea específica. En esta ventana, se muestra el nombre de la tarea, su descripción, estado, prioridad y fecha asociada a la tarea.

Campos:

- **`actual`**: Un objeto de tipo **`Tarea`** que representa la tarea actualmente seleccionada. Esta tarea será la que se muestre en la ventana.

Constructor:

- **`DetallesTareaPropia()`**: Constructor que inicializa los componentes de la ventana y establece las imágenes, configurando así la interfaz para mostrar los detalles de la tarea seleccionada.

Métodos:

- **`main(String[] args)`**: Método principal que sirve como punto de entrada de la aplicación. Aquí es donde se inicializa la ventana y se muestra al usuario.
- **`setTarea(Tarea aux)`**: Establece la tarea que se desea mostrar. El parámetro **`aux`** es un objeto de tipo **`Tarea`** que contiene la información de la tarea seleccionada, como su nombre, descripción, estado, etc.

Herencia:

La clase **`DetallesTareaPropia`** hereda de **`JFrame`**, lo que le permite comportarse como una ventana gráfica en la aplicación. Además, hereda métodos de las clases superiores **`Window`** y **`Component`**, lo que le permite gestionar eventos y manipular la interfaz de usuario.

1.6.Clase `EditarNivelUsuario`

La clase **`EditarNivelUsuario`** extiende `JFrame` y permite editar el nivel de un usuario dentro de la aplicación.

Campos:

- **`indice`**: El índice del usuario en la lista de usuarios.
- **`lista`**: La lista de usuarios cargados desde un archivo, que permite acceder y modificar los datos de los usuarios.
- **`UsuarioActual`**: El objeto que representa al usuario cuya información se está editando.

Constructores:

- **`EditarNivelUsuario(Usuarios usuarios)`**: Inicializa la interfaz gráfica y establece la instancia de la clase `Usuarios`, la cual maneja la lista de usuarios.

Métodos:

- **`setIndice(int ind)`**: Establece el índice del usuario a editar en la lista de usuarios.
- **`setUsuario(Usuario usuario)`**: Establece los datos del usuario que se editarán y los muestra en los componentes gráficos.
- **`setlista(ArrayList<Usuario> aux)`**: Establece la lista de usuarios, generalmente utilizada para cargar los usuarios desde un archivo.

Método `main`:

- **`main(String[] args)`**: Es el punto de entrada principal para ejecutar la ventana de la clase `EditarNivelUsuario`.

1.7.Clase EditarPerfil

- **Extiende**: `JFrame`
- **Interfaz implementada**: `ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`, `RootPaneContainer`, `WindowConstants`.

Descripción:

La clase **EditarPerfil** representa una ventana que permite a los usuarios editar su perfil. Los usuarios pueden modificar su nombre de usuario, contraseña y correo electrónico. La clase hereda de `JFrame`, lo que le permite funcionar como una ventana estándar de una aplicación de escritorio.

Constructores:

- **EditarPerfil()**: Constructor de la clase que inicializa los componentes de la interfaz y ajusta las imágenes. También obtiene el usuario actual y muestra sus datos (nombre de usuario y correo) en los campos correspondientes.

Métodos:

- **main(String[] args)**: Método principal que sirve como punto de entrada para la ventana de la aplicación.

1.8.Clase EditarPerfilUsuario

- **Extiende**: `JFrame`
- **Interfaz implementada**: `ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`, `RootPaneContainer`, `WindowConstants`.

Descripción:

La clase **EditarPerfilUsuario** representa una ventana que permite al usuario editar su perfil. A través de esta interfaz, el usuario puede modificar su nombre de usuario, correo electrónico y contraseña.

Constructores:

- **EditarPerfilUsuario()**: Constructor que inicializa la ventana de edición de perfil. Establece el tamaño y la posición de los componentes gráficos, y asigna los valores iniciales a los campos de texto con los datos del usuario actual.

Métodos:

- **main(String[] args)**: Método principal que actúa como punto de entrada para la ventana.

1.9.Clase EditarTarea

- **Extiende**: `JFrame`

- **Interfaz implementada**: `ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`, `RootPaneContainer`, `WindowConstants`.

Descripción:

La clase **EditarTarea** representa una ventana en la que se pueden editar los detalles de una tarea específica. Permite visualizar y modificar los siguientes campos:

- Nombre
- Descripción
- Fecha
- Estado
- Prioridad
- Usuario asignado a la tarea.

Atributos:

- **indice**: Índice utilizado para navegar a través de la lista de tareas o usuarios.
- **listaBuscar**: Lista de tareas utilizadas para realizar búsquedas.
- **listaUsuarios**: Lista de usuarios registrados en el sistema.
- **tareasActuales**: Objeto que representa la ventana de tareas actuales.
- **userHome**: Directorio base del usuario, apuntando al escritorio.
- **usuario**: Instancia de un usuario con datos inicializados.

Constructores:

- **EditarTarea(TareasActuales tareasActuales)**: Constructor que inicializa la interfaz gráfica y carga los datos necesarios desde el objeto **TareasActuales**, que representa la vista de tareas actuales.

Métodos:

- **listaTareas(ArrayList<Tarea> list)**: Establece la lista de tareas a editar.
  - **Parámetros**: `list` (Lista de tareas a asignar para su visualización y edición).
- **setIndice(int i)**: Establece el índice de la tarea a editar dentro de la lista de tareas.
  - **Parámetros**: `i` (Índice de la tarea a editar).
- **TareaActual()**: Muestra los datos de la tarea seleccionada en los campos correspondientes.
- **main(String[] args)**: Método principal y punto de entrada de la ventana.

1.10.Clase: EditarTareaCompletada

Esta clase representa una ventana de la interfaz gráfica (JFrame) que permite a los usuarios editar tareas que ya han sido completadas. A través de esta ventana, el usuario puede visualizar y modificar detalles de una tarea, como el nombre, descripción, fecha, estado y prioridad.

Campos:

- **usuario**: Instancia del usuario con datos inicializados.
- **userHome**: Directorio base del usuario (generalmente el escritorio).
- **listaBuscar**: Lista de tareas usadas para realizar búsquedas.
- **indice**: Índice para navegar entre tareas o usuarios.
- **historial**: Objeto que representa la ventana del historial de tareas.

Constructor:

- **EditarTareaCompletada(Historial historial)**: Constructor de la clase, inicializa la ventana de edición de tareas y establece los valores iniciales, como la ruta de las tareas y la lista de tareas completadas.

Métodos:

- **listaTareas(ArrayList<Tarea> list)**: Establece la lista de tareas a editar.
- **setIndice(int i)**: Establece el índice de la tarea actual y carga sus detalles.
- **TareaActual()**: Muestra los detalles de la tarea actual en la interfaz gráfica, como nombre, descripción, fecha, estado y prioridad.
- **main(String[] args)**: Punto de entrada principal para ejecutar la ventana.

1.11.Clase `EditarTareaCompletadaUsuario`

Descripción: Es una ventana (JFrame) que permite editar una tarea ya completada, asignada a un usuario. Se usa para modificar el estado de tareas completadas.

Campos:

- `usuario`: Objeto que contiene la información del usuario que está interactuando con la aplicación.
- `userHome`: Ruta base del sistema de archivos donde se almacenan los datos del usuario.
- `listaBuscar`: Lista de tareas obtenidas que están disponibles para ser editadas.
- `indice`: Índice de la tarea seleccionada dentro de la lista de tareas.
- `historialUsuario`: Objeto que representa el historial del usuario desde el cual se abre la ventana.

Constructor:

- **`EditarTareaCompletadaUsuario(HistorialUsuario historial)`**: Inicializa los componentes de la interfaz, asigna el historial del usuario y posiciona la ventana en el centro de la pantalla.

Métodos:

- **`listaTareas(ArrayList<Tarea> list)`**: Establece la lista de tareas disponibles para ser mostradas o editadas.
- **`setIndice(int i)`**: Establece el índice de la tarea actual que se debe mostrar en la interfaz.
- **`TareaActual()`**: Actualiza la interfaz con la información de la tarea seleccionada, incluyendo nombre, descripción, fecha, estado y prioridad.
- **`main(String[] args)`**: Método principal para ejecutar la ventana.

1.12.Clase EditarTareaPropia

Descripción:
La clase `EditarTareaPropia` representa una ventana de la interfaz gráfica para editar tareas de un usuario en la aplicación. Esta ventana permite al usuario modificar y visualizar detalles de las tareas que ha creado.

Campos:

- `indice`: Índice que indica la posición actual de la tarea seleccionada.
- `listaBuscar`: Lista de tareas encontradas en una búsqueda.
- `misTareas`: Objeto que representa las tareas del usuario actual.
- `userHome`: Ruta al directorio principal del usuario en el sistema operativo.
- `usuario`: Objeto que representa al usuario actual.

Constructor:

- `EditarTareaPropia(MisTareas misTareas)`: Constructor que inicializa la interfaz gráfica y configura la ventana, los componentes y las rutas necesarias para manejar las tareas.

Métodos:

- `setDirec()`: Establece las rutas de las tareas para el usuario actual.
- `listaTareas(ArrayList<Tarea> list)`: Establece la lista de tareas desde una lista proporcionada.
- `setIndice(int i)`: Establece el índice de la tarea seleccionada en la lista.
- `TareaActual()`: Muestra los detalles de la tarea actual en los campos de la interfaz.
- `main(String[] args)`: Punto de entrada principal de la ventana.

1.13.Clase `EditarTareaPropiaUsuario`

Descripción:
Esta clase crea una ventana de interfaz gráfica que permite al usuario editar las tareas que ha creado. Se extiende de `JFrame`, lo que la hace una ventana con todas las funcionalidades básicas de un marco de interfaz gráfica en Java.

Campos Principales:

- **`usuario`**: Objeto que representa al usuario que está editando la tarea.
- **`userHome`**: Ruta al directorio principal del usuario en el sistema operativo.
- **`listaBuscar`**: Lista que contiene las tareas encontradas en la búsqueda.
- **`indice`**: Índice que marca la posición de la tarea seleccionada.
- **`misTareasUsuario`**: Objeto de la clase `MisTareasUsuario` que llama a esta ventana.

Constructor:

- **`EditarTareaPropiaUsuario(MisTareasUsuario misTareasUsuario)`**: Inicializa la ventana, ajusta los componentes gráficos, centra la ventana, configura las rutas de las tareas y obtiene el usuario activo.

Métodos:

- **`setDirec()`**: Establece las rutas y la lista de tareas del usuario actual.
- **`listaTareas(ArrayList<Tarea> list)`**: Asigna la lista de tareas a mostrar en la interfaz.
- **`setIndice(int i)`**: Establece el índice de la tarea seleccionada y muestra sus detalles en la interfaz.
- **`TareaActual()`**: Muestra los detalles de la tarea seleccionada (nombre, descripción, día, mes, año, estado y prioridad).
- **`main(String[] args)`**: Método principal que lanza la ventana.

1.14.Clase EditarTareaUsuario

Descripción:
`EditarTareaUsuario` es una ventana gráfica (extiende `JFrame`) utilizada para editar las tareas de un usuario en la aplicación. Permite modificar el estado y otros detalles de las tareas.

Campos:

- **usuario**: Instancia del usuario actual.
- **userHome**: Directorio de inicio del usuario (apunta al escritorio).
- **listaBuscar**: Lista de tareas que el usuario puede visualizar o editar.
- **indice**: Índice de la tarea seleccionada en la lista de tareas.
- **tareasActuales**: Objeto que contiene las tareas actuales del usuario.

Constructores:

- **EditarTareaUsuario(TareasActualesUsuario tareasActuales)**:
  Constructor de la clase. Inicializa los componentes gráficos, centra la ventana y establece la imagen de fondo. Recibe como parámetro un objeto de la clase `TareasActualesUsuario` que gestiona las tareas.

Métodos:

- **listaTareas(ArrayList<Tarea> list)**:
  Asigna una lista de tareas (`list`) a la variable `listaBuscar` para su visualización o edición.
- **setIndice(int i)**:
  Establece el índice de la tarea seleccionada y actualiza la vista de la tarea correspondiente.
- **TareaActual()**:
  Actualiza los campos de la interfaz con los datos de la tarea seleccionada (por ejemplo, nombre, descripción, fecha, etc.).
- **main(String[] args)**:
  Punto de entrada principal para ejecutar la ventana.

1.15.Clase Escalar

Descripción:
La clase `Escalar` proporciona métodos para redimensionar imágenes y ajustarlas a un componente gráfico, como un `JLabel`. Esto permite que las imágenes se escalen automáticamente para adaptarse al tamaño del componente de destino.

Constructor:

- **Escalar()**:
  Constructor predeterminado de la clase `Escalar`. No realiza ninguna acción específica en este caso.

Métodos:

- **escalarLabel(JLabel label, String rutaImagen)**:
  Redimensiona una imagen y la ajusta a las dimensiones de un `JLabel`. Toma la imagen desde la ruta proporcionada, la escala según el tamaño del `JLabel` y la establece como el icono de dicho `JLabel`.
  Parámetros:

- **label**: El `JLabel` cuyo tamaño se usará para redimensionar la imagen.
- **rutaImagen**: La ruta del archivo de imagen que se debe cargar y redimensionar (relativa al paquete de recursos del proyecto).

1.16.Clase Historial

Descripción:
La clase `Historial` representa una ventana que muestra el historial de tareas de un usuario. Proporciona funcionalidades para buscar tareas y mostrar sus detalles en una interfaz gráfica.

Campos (Atributos)

- **cont**:
  Contador utilizado para realizar un seguimiento de las tareas o resultados en el historial.
- **estado**:
  Estado utilizado para filtrar tareas durante la búsqueda.
- **indice**:
  Índice actual en la lista de tareas.
- **lista**:
  Lista de todas las tareas que el usuario ha creado o que están asignadas al usuario.
- **listaBuscar**:
  Lista utilizada para almacenar las tareas filtradas o aquellas que cumplen con ciertos criterios de búsqueda.
- **nombre**:
  Nombre utilizado para filtrar tareas durante la búsqueda.
- **prioridad**:
  Prioridad utilizada para filtrar tareas durante la búsqueda.

Constructor

- **Historial()**:
  Constructor de la clase `Historial`. Inicializa los componentes de la interfaz gráfica, ajusta las imágenes en los labels utilizando la clase `Escalar`, obtiene el usuario actual, carga las tareas completadas y las muestra en la interfaz.

Métodos

- **setLista()**:
  Establece la lista de tareas, filtrando solo aquellas que están completadas. Limpia la tabla actual y vuelve a cargar la lista de tareas filtrada, o ejecuta una búsqueda si los filtros de nombre, prioridad o estado están configurados.
- **vaciarTabla()**:
  Vacía la tabla eliminando todas las filas. Recorre las filas de la tabla de la última a la primera y las elimina.
- **filtrarTareas(String textoBusqueda, String prioridadSeleccionada, String estadoSeleccionado, ArrayList<Tarea> lista)**:
  Filtra las tareas de la lista según los criterios proporcionados: texto de búsqueda, prioridad seleccionada y estado seleccionado.

  Parámetros:
  - **textoBusqueda**: Texto utilizado para buscar tareas por nombre.
  - **prioridadSeleccionada**: Prioridad seleccionada para filtrar tareas.
  - **estadoSeleccionado**: Estado seleccionado para filtrar tareas.
  - **lista**: Lista de tareas a filtrar.
- **Retorna**:
  Una nueva lista de tareas que coinciden con los criterios de búsqueda.

- **buscarTarea()**:
  Realiza una búsqueda de tareas aplicando los filtros de nombre, prioridad y estado seleccionados. Luego, actualiza la tabla con los resultados de la búsqueda.
- **main(String[] args)**:
  Método principal que es el punto de entrada de la ventana.

1.17.Clase HistorialUsuario

Descripción: La clase `HistorialUsuario` extiende `JFrame` y representa una ventana en la que se muestra el historial de tareas de un usuario. Permite visualizar, buscar y filtrar tareas basadas en criterios como nombre, prioridad y estado. Además, gestiona la carga de tareas de un usuario específico y proporciona métodos para manipular estas tareas y su visualización.

Campos:

- **`cont`**: Contador para realizar un seguimiento de las tareas o resultados en el historial.
- **`estado`**: Filtro para buscar tareas por estado.
- **`indice`**: Índice de la tarea seleccionada en la lista de tareas.
- **`lista`**: Lista de todas las tareas del usuario.
- **`listaBuscar`**: Lista para almacenar tareas filtradas según criterios de búsqueda.
- **`nombre`**: Filtro para buscar tareas por nombre.
- **`prioridad`**: Filtro para buscar tareas por prioridad.

Constructor:

- **`HistorialUsuario()`**: Inicializa los componentes de la interfaz gráfica, ajusta imágenes, carga el usuario actual y filtra las tareas completadas para mostrarlas en la tabla de historial.

Métodos:

1. **`setLista()`**:
   - Establece la lista de tareas, filtrando las completadas. Luego, actualiza la tabla con las tareas filtradas o realiza una búsqueda según los filtros aplicados.
2. **`vaciarTabla()`**:
   - Vacía todas las filas de la tabla de tareas en la interfaz de usuario.
3. **`filtrarTareas(String textoBusqueda, String prioridadSeleccionada, String estadoSeleccionado, ArrayList<Tarea> lista)`**:
   - Filtra las tareas según los filtros de nombre, prioridad y estado. Retorna una lista con las tareas que coinciden con los criterios.
4. **`buscarTarea()`**:
   - Realiza una búsqueda de tareas aplicando filtros de nombre y prioridad. Actualiza la tabla con los resultados de la búsqueda o muestra un mensaje de error si no se encuentran tareas.
5. **`main(String[] args)`**:
   - Punto de entrada principal de la aplicación, donde se inicializa la ventana.

1.18.Clase Inicio

Descripción: La clase `Inicio` extiende `JFrame` y representa la ventana principal de la aplicación. Esta clase es responsable de gestionar la interfaz gráfica inicial de la aplicación y configurar propiedades relacionadas con las tareas y el usuario.

Campos:

- **`lista`**: Es una lista de tareas que la aplicación maneja. Almacena todas las tareas gestionadas por el sistema.

Constructor:

- **`Inicio()`** : Inicializa los componentes de la interfaz gráfica y configura diversas propiedades relacionadas con el usuario y las tareas.

Métodos:

1. **`main(String[] args)`**:
    - Es el punto de entrada principal de la aplicación, donde se inicializa la ventana. Recibe los argumentos de la línea de comandos si los hay.

1.19.Clase InicioUsuario

Descripción: La clase `InicioUsuario` extiende `JFrame` y representa la ventana principal de la interfaz de usuario de la aplicación. Se encarga de gestionar los componentes gráficos y mostrar la información relevante sobre el usuario y sus tareas.

Campos:

- **`lista`**:
    - Tipo: `ArrayList<Tarea>`
    - Descripción: Lista que contiene las tareas que la aplicación maneja.

Constructores:

- **`InicioUsuario()`**:
    - Descripción: Constructor que inicializa los componentes gráficos de la interfaz. También establece las imágenes en las etiquetas, obtiene el usuario actual, carga la lista de tareas y calcula el número de tareas en diferentes estados.

Métodos:

1. **`main(String[] args)`**:
    - Descripción: Es el punto de entrada principal de la ventana. Se ejecuta cuando se inicia la aplicación.
    - Parámetros:
        - `args` (String[]): Argumentos de la línea de comandos.

1.20.Clase `Login`

Descripción: La clase `Login` extiende `JFrame` y representa la ventana de inicio de sesión de la aplicación. Se encarga de gestionar la interfaz gráfica para que los usuarios puedan ingresar sus credenciales, así como la verificación de la existencia de los archivos necesarios para la operación de la aplicación.

Campos:

- **`lista`**:

- Tipo: `ArrayList<Usuario>`
- Descripción: Lista de usuarios registrados en la aplicación.

Constructores:

- **`Login()`**:
  - Descripción: Constructor de la clase `Login`. Inicializa los componentes gráficos de la interfaz, verifica si los archivos necesarios existen y, si no, los crea. Además, carga la lista de usuarios desde el archivo correspondiente.

Métodos:

1. **`main(String[] args)`**:
   - Descripción: Punto de entrada principal de la ventana de inicio de sesión.
   - Parámetros:
     - `args` (String[]): Argumentos de la línea de comandos.
2. **`creadorArchivos(String userHome)`**:
   - Descripción: Método encargado de crear la estructura de carpetas y archivos necesarios para la aplicación. Si las carpetas y archivos no existen, los crea y escribe un usuario predeterminado en el archivo `Usuarios.txt`. Las carpetas creadas son `Taskflow` y `Tareas`, y los archivos creados son `Usuarios.txt`, `UsuarioActual.txt` y `ListaTareas.txt`.
   - Parámetros:
     - `userHome` (String): La ruta del directorio principal del usuario (por lo general, la ruta de inicio del usuario en el sistema).

1.21.Clase MisTareas

Descripción:
`MisTareas` es una clase que representa una ventana donde se gestionan y visualizan las tareas asignadas al usuario actual. Extiende de `JFrame` y permite filtrar y buscar tareas por diferentes criterios (nombre, prioridad, estado).

Campos (Fields)

- **`cont`**: Contador general de tareas, usado para controlar o verificar el número de tareas.
- **`estado`**: Estado de las tareas, utilizado como filtro de búsqueda.
- **`indice`**: Índice de la tarea actual en la lista.
- **`lista`**: Lista de todas las tareas del usuario.
- **`listaBuscar`**: Lista de tareas filtradas según los criterios de búsqueda.
- **`nombre`**: Nombre de la tarea, usado como criterio de búsqueda.
- **`prioridad`**: Prioridad de las tareas, utilizada para filtrar tareas.

Constructor

- **`MisTareas()`** : Inicializa la interfaz de usuario, ajusta las imágenes de los elementos visuales, y carga las tareas del usuario, filtrando las que no están completas. Luego, muestra estas tareas en una tabla.

Métodos

- **`setLista()`** : Actualiza la lista de tareas obteniéndolas desde un archivo y filtrando las incompletas. Después llena la tabla con las tareas filtradas o realiza una búsqueda si el usuario ha aplicado filtros.

- **`vaciarTabla()`**: Elimina todas las filas de la tabla, desde la última hasta la primera.
- **`filtrarTareas(String textoBusqueda, String prioridadSeleccionada, String estadoSeleccionado, ArrayList<Tarea> lista)`**: Filtra las tareas de la lista según el nombre, la prioridad y el estado. Devuelve una nueva lista con las tareas que coinciden con los filtros aplicados.
- **`buscarTarea()`**: Realiza una búsqueda de tareas usando los filtros establecidos por el usuario (nombre, prioridad, estado) y muestra los resultados en una tabla. Si no hay resultados, muestra un mensaje de advertencia.
- **`main(String[] args)`**: Método principal que inicia la ventana de la aplicación.

1.22. Clase `MisTareasUsuario`

Hereda de **`JFrame`**, lo que significa que es una ventana (frame) dentro de una aplicación de escritorio Java, específicamente usando la biblioteca Swing.

Atributos de la Clase

- **`cont`**: Contador que lleva el seguimiento del número total de tareas asociadas al usuario.
- **`estado`**: Guarda el estado actual de la tarea, como "pendiente", "en progreso", "completada", etc.
- **`indice`**: Índice de la tarea seleccionada en la lista, usado para realizar operaciones en la tarea seleccionada.
- **`lista`**: Es una lista que contiene todas las tareas asociadas al usuario. Cada tarea está representada por un objeto de la clase `Tarea`.
- **`listaBuscar`**: Lista auxiliar utilizada para almacenar tareas que coinciden con los criterios de búsqueda.
- **`nombre`**: El nombre de la tarea o del usuario. Se usa para filtrar las tareas por nombre o para representar el nombre del usuario.
- **`prioridad`**: La prioridad asignada a una tarea (puede ser algo como "alta", "media" o "baja").
- **`estado`**: El estado de la tarea (ej. "pendiente", "en progreso", "completada").

Métodos Principales

1. `MisTareasUsuario()`:

Constructor de la clase. Inicializa los componentes de la interfaz gráfica de usuario y configura la lista de tareas del usuario, probablemente cargando las tareas desde un archivo o base de datos.

2. `setLista()`:

Este método actualiza la lista de tareas. Carga las tareas desde un archivo (probablemente guardadas previamente) y filtra aquellas que están marcadas como "Completada". Luego, la interfaz de usuario (como una tabla o lista) es actualizada con la nueva lista de tareas.

3. `vaciarTabla()`:

Este método limpia la tabla de tareas, eliminando todas las filas de la interfaz gráfica. Es útil cuando se necesita mostrar una nueva lista de tareas después de filtrar o actualizar los datos.

4. `filtrarTareas(String textoBusqueda, String prioridadSeleccionada, String estadoSeleccionado, ArrayList<Tarea> lista)`:

Este método permite filtrar las tareas según el texto de búsqueda (nombre de la tarea), la prioridad seleccionada y el estado de la tarea (como "pendiente", "completada", etc.). Devuelve una lista de tareas que cumplen con los criterios de búsqueda proporcionados.

- **Parámetros**:

- `textoBusqueda`: Texto que se usará para buscar por nombre de tarea.
- `prioridadSeleccionada`: Filtro de prioridad de la tarea.
- `estadoSeleccionado`: Filtro de estado de la tarea (ej., "pendiente", "completada").
- `lista`: Lista de tareas a filtrar.

- **Devuelve**: Una lista de tareas que cumplen los filtros aplicados.

5. `buscarTarea()`:

Ejecuta la búsqueda de tareas basándose en los filtros seleccionados por el usuario (por nombre, prioridad y estado). Después de realizar la búsqueda, la interfaz gráfica (tabla o lista) se actualiza con los resultados. Si no se encuentran tareas que coincidan con los filtros, se puede mostrar un mensaje de advertencia.

6. `main(String[] args)`:

Es el punto de entrada principal de la ventana. Inicializa y muestra la interfaz de usuario para que el usuario interactúe con las tareas.

Funcionalidad General

- La clase **`MisTareasUsuario`** se centra en proporcionar una interfaz para gestionar tareas de manera visual. Permite realizar filtrados de tareas por diferentes criterios (nombre, prioridad, estado) y actualizar dinámicamente la vista de las tareas a través de una tabla o lista.
- **`setLista()`** es responsable de cargar y actualizar la lista de tareas desde un archivo o base de datos. Además, filtra las tareas "completadas" para que no se muestren si así lo indica la lógica de la aplicación.
- **`vaciarTabla()`** es útil cuando se necesitan realizar cambios en la tabla o lista de tareas, como eliminar todas las filas antes de mostrar una nueva lista después de una búsqueda o actualización.
- El método **`filtrarTareas()`** realiza un filtrado dinámico de tareas basado en los criterios que el usuario elige (nombre, prioridad y estado), lo que permite a los usuarios encontrar rápidamente las tareas que les interesan.

1.23.Clase PanelRound

La clase `PanelRound` es una subclase de `JPanel` que permite crear paneles con **esquinas redondeadas personalizables**. Cada esquina puede tener un radio de redondeo independiente, lo que proporciona flexibilidad en su diseño.

Características principales:

- **Herencia**: Extiende de `JPanel`, por lo que hereda todas las funcionalidades de un panel estándar de Swing, como gestión de eventos, gráficos, etc.
- **Bordes redondeados**: La principal característica de `PanelRound` es la capacidad de redondear las esquinas del panel. El radio de redondeo puede ajustarse para cada esquina de forma independiente.

Métodos importantes:

- **Obtener valores de redondeo**:
  - `getRoundTopLeft()`: Obtiene el radio de redondeo de la esquina superior izquierda.
  - `getRoundTopRight()`: Obtiene el radio de redondeo de la esquina superior derecha.
  - `getRoundBottomLeft()`: Obtiene el radio de redondeo de la esquina inferior izquierda.
  - `getRoundBottomRight()`: Obtiene el radio de redondeo de la esquina inferior derecha.
- **Establecer valores de redondeo**:

    - `setRoundTopLeft(int roundTopLeft)`: Establece el radio de redondeo para la esquina superior izquierda.
    - `setRoundTopRight(int roundTopRight)`: Establece el radio de redondeo para la esquina superior derecha.
    - `setRoundBottomLeft(int roundBottomLeft)`: Establece el radio de redondeo para la esquina inferior izquierda.
    - `setRoundBottomRight(int roundBottomRight)`: Establece el radio de redondeo para la esquina inferior derecha.
- **Dibujo del panel**:
    - `paintComponent(Graphics grphcs)`: Sobrescribe el método `paintComponent` de `JComponent` para dibujar el panel con las esquinas redondeadas de acuerdo con los valores establecidos.

Constructor:

- `PanelRound()`: El constructor inicializa el panel y establece su propiedad de opacidad a `false`, lo que permite que el fondo se vea a través del panel si es necesario.

1.24.Clase Perfil

La clase `Perfil` extiende de `JFrame` y se utiliza para mostrar el perfil de un usuario en la interfaz gráfica. Esta ventana contiene componentes gráficos que presentan la información del usuario, como su nombre, foto, etc.

Características principales:

- **Herencia**:
    - Extiende de `JFrame`, lo que significa que se comporta como una ventana estándar en una aplicación de escritorio Swing.
    - Hereda funcionalidades como el manejo de eventos, componentes gráficos y opciones de configuración de la ventana (tamaño, ubicación, etc.).

Constructor:

- **`Perfil()`**:
    - Inicializa los componentes de la interfaz gráfica.
    - Ajusta la ubicación de la ventana.
    - Escala las imágenes mostradas en los `JLabel`.
    - Recupera los datos del usuario actual y los muestra en los `JLabel` correspondientes.

Métodos importantes:

- **`main(String[] args)`**:
    - Este es el punto de entrada principal de la ventana. Es el método que arranca la aplicación.

Métodos heredados:

- La clase hereda varios métodos de `JFrame`, `Frame` y `Window`, que permiten realizar operaciones comunes en ventanas, como:
    - **Configuración de la ventana**: Cambiar tamaño, visibilidad, íconos, título, etc.
    - **Eventos de la ventana**: Manejo de eventos como cierre de ventana o cambios de estado.
    - **Composición y diseño**: Añadir componentes, cambiar el layout, y gestionar los eventos de entrada (teclado, ratón).

1.25.Clase `PerfilUsuario`

Descripción:
La clase `PerfilUsuario` representa una ventana en la interfaz gráfica de un programa que muestra el perfil de un usuario. Hereda de `JFrame`, lo que le permite manejar características típicas de una ventana en una aplicación gráfica. Su función principal es visualizar los datos del usuario, incluyendo información almacenada en etiquetas (`JLabel`).

Constructor:

- `PerfilUsuario()`:
  Inicializa los componentes de la interfaz gráfica, ajusta la ubicación de la ventana, escala las imágenes mostradas en los `JLabel`, y obtiene los datos del usuario para mostrarlos en las etiquetas correspondientes.

Métodos Principales:

- `main(String[] args)`:
  Método principal que sirve como punto de entrada para ejecutar la ventana de perfil del usuario.

Herencia:

La clase extiende `JFrame`, lo que le permite acceder a todas las funcionalidades y comportamientos de un marco de ventana estándar. También implementa diversas interfaces, como `ImageObserver`, `MenuContainer`, `Serializable`, entre otras.

Métodos heredados:

- De `JFrame`: Métodos como `setDefaultCloseOperation()`, `getContentPane()`, `setJMenuBar()`, `repaint()`, entre otros, para manipular la ventana, su contenido y eventos.
- De `Window`: Métodos para manejar eventos relacionados con la ventana, como `setOpacity()`, `setVisible()`, `setSize()`, etc.
- De `Container`: Métodos para manejar componentes dentro del contenedor, como `add()`, `remove()`, `getLayout()`, entre otros.

1.26.Clase `PerfilView`

Descripción:
La clase `PerfilView` representa la vista del perfil de un usuario dentro de la interfaz gráfica de una aplicación. Permite visualizar y modificar la información relacionada con el usuario actual. Esta clase también se encarga de la gestión visual de los componentes gráficos, utilizando la clase `Escalar` para ajustar elementos visuales como imágenes en la interfaz.

Constructor:

- `PerfilView()`:
  Inicializa los componentes gráficos de la interfaz, ajusta la ubicación de la ventana, escala las imágenes mostradas en los `JLabel`, y muestra la información del usuario actual.

Métodos Principales:

- `setUsuario(Usuario usuario)`:
  Actualiza la interfaz con la información de un usuario específico. Modifica las etiquetas y otros componentes visuales para reflejar los datos del objeto `Usuario` proporcionado.
- `main(String[] args)`:
  Método principal que actúa como punto de entrada para ejecutar la ventana del perfil de usuario.

Atributos:

- `UsuarioActual`:
  Una instancia del objeto `Usuario` que representa al usuario actualmente activo o cuyo perfil está siendo visualizado.

Herencia:

La clase `PerfilView` extiende `JFrame`, lo que le otorga todas las funcionalidades y comportamientos de una ventana estándar. También implementa interfaces como `ImageObserver`, `MenuContainer`, `Serializable`, entre otras.

Métodos heredados:

- De `JFrame`: Métodos como `setDefaultCloseOperation()`, `getContentPane()`, `setJMenuBar()`, `repaint()`, entre otros, que permiten manejar la ventana, el contenido y eventos asociados.
- De `Window`: Métodos relacionados con el control de la ventana, como `setOpacity()`, `setSize()`, `setVisible()`, etc.
- De `Container`: Métodos para manejar y organizar los componentes dentro de la ventana, como `add()`, `remove()`, `getLayout()`, etc.

1.27.Clase `Registro`

Descripción:
La clase `Registro` representa la ventana de registro de un usuario en la interfaz gráfica de una aplicación. Su función principal es permitir la creación de un nuevo usuario mediante un formulario, además de validar los datos ingresados antes de crear la cuenta. La clase maneja la interacción con el usuario y la validación de los campos del formulario de registro.

Constructor:

- **`Registro()`**:
  Este constructor inicializa los componentes gráficos de la interfaz, ajusta la ubicación de la ventana al centro de la pantalla y escala las imágenes mostradas en los `JLabel`.

Métodos Principales:

- **`main(String[] args)`**:
  Método principal que actúa como punto de entrada para ejecutar la ventana de registro de usuario.

Herencia:

La clase `Registro` extiende `JFrame`, lo que significa que hereda funcionalidades de una ventana estándar en una aplicación Swing. Además, implementa interfaces como `ImageObserver`, `MenuContainer`, `Serializable`, y otras relacionadas con la accesibilidad y el manejo de ventanas.

Métodos heredados:

- **De `JFrame`**:
  Métodos como `setDefaultCloseOperation()`, `getContentPane()`, `setJMenuBar()`, `repaint()`, entre otros, para gestionar la ventana y su comportamiento.
- **De `Window`**:
  Métodos para manejar las propiedades de la ventana, como `setOpacity()`, `setVisible()`, `setSize()`, etc.
- **De `Container`**:
  Métodos para gestionar los componentes dentro de la ventana, como `add()`, `remove()`, `setLayout()`, entre otros.
- **De `Component`**:
  Métodos como `setFocusable()`, `setSize()`, `setFont()`, etc., que permiten interactuar con los componentes visuales en la ventana.

1.28.Clase `Tarea`

Descripción:
La clase `Tarea` representa una tarea en un sistema de gestión de tareas. Cada tarea tiene un conjunto de atributos, como número de tarea, nombre, descripción, prioridad, estado, comentario, usuario asignado y fecha de entrega. Esta clase proporciona métodos para crear, modificar, eliminar, guardar y obtener información de tareas, así como para gestionar los archivos asociados a las tareas y calcular la fecha de entrega.

Atributos:

- **`numTarea`**: Número único que identifica la tarea.
- **`nombre`**: Nombre de la tarea.
- **`descripcion`**: Descripción de la tarea.
- **`prioridad`**: Prioridad de la tarea (por ejemplo, Alta, Media, Baja).
- **`estado`**: Estado de la tarea, que puede ser "Por hacer", "En progreso", "Completada".
- **`comentario`**: Comentarios asociados a la tarea.
- **`usuarioAsignado`**: Nombre del usuario asignado a la tarea.
- **`dia`**: Día de la fecha de entrega de la tarea.
- **`mes`**: Mes de la fecha de entrega de la tarea.
- **`year`**: Año de la fecha de entrega de la tarea.

Constructores:

1. **`Tarea()`**: Constructor por defecto, inicializa una tarea sin valores predefinidos.
2. **`Tarea(int NumeroTarea, String Nombre, String Descripcion, String Prioridad, String UsuarioAsignado, int dia, String mes)`**: Constructor que inicializa una tarea con todos los detalles excepto el año.
3. **`Tarea(int NumeroTarea, String Nombre, String Descripcion, String Prioridad, String UsuarioAsignado, int dia, String mes, int year)`**: Constructor que inicializa una tarea completa con todos los detalles, incluida la fecha de entrega.
4. **`Tarea(int NumeroTarea, String Nombre, String Descripcion, String Prioridad, String Estado, String UsuarioAsignado, int dia, String mes)`**: Constructor que también incluye el estado de la tarea.
5. **`Tarea(Tarea aux)`**: Constructor de copia que crea una nueva tarea basada en una tarea existente.

Métodos Principales:

- **`CrearNuevaTarea(String direccion, Tarea act)`**: Crea un archivo para una nueva tarea en la ubicación especificada.
- **`DetallesTarea(String direccion, int NumeroTarea)`**: Obtiene los detalles de una tarea a partir de su archivo.
- **`editarTarea(String direccion, Tarea act, String listaTareas, Tarea ant)`**: Modifica una tarea, tanto en su archivo individual como en el archivo de lista de tareas.
- **`eliminarTarea(String direccion, Tarea tarea, String listaTareas, String direc)`**: Elimina una tarea tanto del archivo individual como de la lista de tareas.
- **`guardarTarea(String direccion, Tarea act, String listaTareas)`**: Guarda una tarea en su archivo correspondiente y en el archivo de lista de tareas.
- **`tareasTotal(String direccion)`**: Obtiene el número total de tareas en el archivo de lista de tareas.
- **`getFechaEntrega()`**: Obtiene la fecha de entrega de la tarea en formato `Date`.
- **`ListaTareas(String direc)`**: Obtiene una lista de todas las tareas presentes en el archivo de lista de tareas.
- **`numeroNuevaTarea(String direccion)`**: Obtiene el número de la siguiente tarea disponible para crear.

Métodos de acceso y modificación (Getters y Setters):

- **`getComentario()`**: Obtiene el comentario asociado a la tarea.
- **`getDescripcion()`**: Obtiene la descripción de la tarea.
- **`getDia()`**: Obtiene el día de la tarea.
- **`getEstado()`**: Obtiene el estado de la tarea.
- **`getMes()`**: Obtiene el mes de la tarea.
- **`getNombre()`**: Obtiene el nombre de la tarea.
- **`getNumeroTarea()`**: Obtiene el número de la tarea.
- **`getPrioridad()`**: Obtiene la prioridad de la tarea.
- **`getUsuarioAsignado()`**: Obtiene el usuario asignado a la tarea.
- **`getYear()`**: Obtiene el año de la tarea.
- **`setComentario(String comentario)`**: Establece el comentario de la tarea.
- **`setDescripcion(String descripcion)`**: Establece la descripción de la tarea.
- **`setDia(int dia)`**: Establece el día en que se debe completar la tarea.
- **`setEstado(String estado)`**: Establece el estado de la tarea.
- **`setMes(String mes)`**: Establece el mes en que se debe completar la tarea.
- **`setNombre(String nombre)`**: Establece el nombre de la tarea.
- **`setNumeroTarea(int numero)`**: Establece el número de la tarea.
- **`setPrioridad(String prioridad)`**: Establece la prioridad de la tarea.
- **`setUsuarioAsignado(String usuarioAsignado)`**: Establece el usuario asignado a la tarea.
- **`setYear(int year)`**: Establece el año en que se debe completar la tarea.

Métodos adicionales:

- **`indiceTarea(String contenido)`**: Convierte una cadena de texto que contiene el número y nombre de la tarea en un objeto `Tarea`.
- **`editarCualquierTarea(String direccionTarea, Tarea act, String listaTareas, Tarea ant, String direc)`**: Permite editar cualquier tarea dentro del sistema, actualizando tanto su archivo individual como el archivo de lista de tareas.

Métodos para manejo de archivos:

- **`ListaTareas(String direc)`**: Obtiene una lista de todas las tareas presentes en el archivo de lista de tareas.
- **`guardarTarea(String direccion, Tarea act, String listaTareas)`**: Guarda la tarea tanto en su archivo individual como en el archivo de lista de tareas.

- **`eliminarTarea(String direccion, Tarea tarea, String listaTareas, String direc)`**: Elimina la tarea de los archivos correspondientes.
- **`CrearNuevaTarea(String direccion, Tarea act)`**: Crea un archivo para una nueva tarea.

1.29.Clase `TareaRenderer`

Descripción general: La clase `TareaRenderer` extiende `DefaultTableCellRenderer` y se utiliza para personalizar la representación de celdas en una tabla de tareas. Su principal función es modificar el color de fondo de las celdas según la prioridad de la tarea o si la tarea está atrasada. Esta clase se usa en aplicaciones donde se necesita resaltar tareas de acuerdo a ciertos criterios, como la prioridad y la fecha de entrega.

Detalles importantes:

- **Herencia**: `TareaRenderer` hereda de `DefaultTableCellRenderer`, que a su vez hereda de clases como `JLabel` y `JComponent`. Esto le permite personalizar la apariencia de las celdas en una tabla (`JTable`).
- **Interfaz implementada**: Implementa la interfaz `TableCellRenderer`, lo que le permite definir cómo se deben renderizar las celdas de una tabla en una GUI.

Constructores:

- **`TareaRenderer(ArrayList<Tarea> lista)`**: Constructor que recibe una lista de tareas asociadas a las filas de la tabla. Esta lista es usada para determinar los colores y estilos de las celdas en función de las propiedades de las tareas.

Métodos:

- **`getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column)`**: Este es el método sobrescrito de `DefaultTableCellRenderer` que permite personalizar la representación de las celdas en función de la tarea asociada a la fila y columna correspondientes.
    - Modifica el color de fondo de la celda en función de:
        - **Prioridad de la tarea**: Se cambia el color de fondo para resaltar la prioridad de la tarea.
        - **Fecha de entrega**: Si la tarea está atrasada, la celda se colorea con un fondo especial.

1.30.Clase `TareaRenderer2`

Descripción general: La clase `TareaRenderer2` es una subclase de `DefaultTableCellRenderer` que se utiliza para personalizar la representación de celdas en una tabla (`JTable`). Su principal función es cambiar el color de fondo de las celdas según la **prioridad** de la tarea asociada a esa fila. A diferencia de otras clases similares, esta versión se centra únicamente en la prioridad de la tarea para determinar el color de la celda.

Detalles importantes:

- **Herencia**:
    - Hereda de `DefaultTableCellRenderer`, lo que le permite personalizar el renderizado de celdas en una tabla.
    - A través de `DefaultTableCellRenderer`, también hereda de clases como `JLabel` y `JComponent`.
- **Interfaz implementada**:
    - Implementa la interfaz `TableCellRenderer`, lo que le permite ser utilizado para renderizar celdas en un `JTable`.

Constructores:

- **`TareaRenderer2(ArrayList<Tarea> lista)`**:
  - Constructor que recibe una lista de tareas asociadas a las filas de la tabla. La lista se utiliza para determinar qué color de fondo asignar a cada celda según la prioridad de la tarea en esa fila.

Métodos:

- **`getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column)`**:
  - Este método sobrescribe el de `DefaultTableCellRenderer` y es responsable de personalizar cómo se muestra cada celda en la tabla.
  - Cambia el color de fondo de la celda dependiendo de la prioridad de la tarea asociada con esa fila.
- **Parámetros**:
  - **`table`**: La tabla donde se está renderizando la celda.
  - **`value`**: El valor mostrado en la celda.
  - **`isSelected`**: Indica si la celda está seleccionada.
  - **`hasFocus`**: Indica si la celda tiene el foco.
  - **`row`**: El índice de la fila donde se encuentra la celda.
  - **`column`**: El índice de la columna donde se encuentra la celda.
- **Retorna**: El componente renderizado que se muestra en la celda.

1.31.Clase `TareasActuales`

La clase `TareasActuales` es una ventana gráfica que gestiona y muestra las tareas actuales de un usuario. Permite filtrar las tareas según diversos criterios como nombre, prioridad, estado y usuario, y facilita la visualización y administración de las tareas en la aplicación Taskflow.

Herencia

- **Hereda de**: `JFrame` (ventana principal de la interfaz gráfica de usuario).
- **Implementa interfaces**: `ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`, `RootPaneContainer`, `WindowConstants`.

Atributos Principales

- **`cont`**: Contador utilizado para manejar las tareas (posiblemente relacionado con la paginación).
- **`estado`**: Filtro para buscar tareas por su estado.
- **`indice`**: Índice utilizado para seleccionar tareas en la lista.
- **`lista`**: Lista de todas las tareas actuales.
- **`listaBuscar`**: Lista de tareas filtradas según los criterios de búsqueda.
- **`listaUsuarios`**: Lista de usuarios registrados en la aplicación.
- **`nombre`, `prioridad`, `usuario`**: Filtros para buscar tareas por nombre, prioridad y usuario.

Constructor

- **`TareasActuales()`**: Inicializa la ventana y los componentes gráficos de la interfaz.

Métodos Principales

- **`buscarTarea()`**: Realiza una búsqueda de tareas basándose en los filtros seleccionados por el usuario y actualiza la visualización.
- **`EnProgreso()`**: Filtra y muestra las tareas que están "En progreso" para el usuario actual.
- **`Completadas()`**: Filtra y muestra las tareas "Completadas" para el usuario actual.
- **`PorHacer()`**: Filtra y muestra las tareas "Por hacer" para el usuario actual.
- **`filtrarTareas()`**: Filtra la lista de tareas según los criterios proporcionados (texto de búsqueda, usuario, prioridad, estado).
- **`setLista()`**: Carga las tareas desde el archivo de tareas y actualiza la lista mostrada en la interfaz, aplicando filtros si es necesario.
- **`vaciarTabla()`**: Limpia la tabla de tareas, eliminando todas las filas.
- **`main(String[] args)`**: Punto de entrada principal de la ventana (ejecución de la aplicación).

Descripción de algunos métodos

- **`filtrarTareas(String textoBusqueda, String usuarioSeleccionado, String prioridadSeleccionada, String estadoSeleccionado, ArrayList<Tarea> lista)`**: Filtra las tareas según el texto de búsqueda, el usuario, la prioridad y el estado seleccionados, devolviendo la lista filtrada.

Comportamiento General

- La clase permite gestionar y visualizar las tareas de forma interactiva, filtrándolas por diversos criterios.
- Los usuarios pueden ver tareas "Por hacer", "En progreso" o "Completadas" según su estado.
- La lista de tareas se actualiza dinámicamente en función de los filtros aplicados.

1.32.Clase: TareasActualesUsuario

Extiende de: `javax.swing.JFrame`
Descripción:
La clase `TareasActualesUsuario` es una ventana gráfica que permite gestionar las tareas de un usuario. Extiende de `JFrame` y maneja operaciones relacionadas con la visualización, filtrado y manipulación de tareas.

Campos

- **`estado`**: Estado de las tareas a filtrar (por ejemplo, "Pendiente", "Completada").
- **`indice`**: Índice de la tarea seleccionada.
- **`lista`**: Lista de todas las tareas asociadas al usuario.
- **`listaBuscar`**: Lista de tareas filtradas.
- **`nombre`**: Nombre del usuario.
- **`prioridad`**: Prioridad de las tareas a filtrar.

Métodos

- **`buscarTarea()`**: Filtra las tareas según los filtros seleccionados (nombre, prioridad, estado).
- **`Completadas()`**: Filtra y muestra las tareas con estado "Completada".
- **`EnProgreso()`**: Filtra y muestra las tareas con estado "En progreso".
- **`PorHacer()`**: Filtra y muestra las tareas con estado "Por hacer".
- **`setLista()`**: Recarga las tareas del usuario y actualiza la tabla con los filtros seleccionados.
- **`filtrarTareas(String textoBusqueda, String prioridadSeleccionada, String estadoSeleccionado, ArrayList<Tarea> lista)`**: Filtra las tareas de acuerdo con los criterios proporcionados.

- **`tareasUsuario()`**: Obtiene las tareas asociadas al usuario desde un archivo.
- **`vaciarTabla()`**: Elimina todas las filas de la tabla.
- **`main(String[] args)`**: Método principal para iniciar la ventana.

Constructor:

- **`TareasActualesUsuario()`**: Inicializa los componentes de la interfaz gráfica, establece el usuario actual y carga las tareas.

Descripción de los filtros en métodos específicos:

- **Filtrar tareas**: Los métodos como `EnProgreso()`, `PorHacer()`, y `Completadas()` filtran las tareas en la tabla según su estado. Cada uno de estos métodos limpia la tabla, aplica el filtro correspondiente y muestra las tareas filtradas.

1.33.Clase: Usuario

Descripción:
La clase `Usuario` representa a un usuario en el sistema. Gestiona su información básica como nombre de usuario, contraseña, correo, nivel de acceso, y foto de perfil. También permite realizar operaciones de registro, edición y eliminación de usuarios, así como gestionar la persistencia de la información en archivos de texto.

Constructores:

- **`Usuario()`**: Constructor por defecto, crea un usuario sin inicializar atributos.
- **`Usuario(int id, String usuario, String contraseña, String correo)`**: Inicializa un nuevo usuario con un ID, nombre de usuario, contraseña y correo.
- **`Usuario(int id, String usuario, String contraseña, String nivel, String correo)`**: Inicializa un usuario con ID, nombre, contraseña, nivel y correo.
- **`Usuario(String usuario, String contraseña, String nivel, String correo)`**: Crea un usuario sin ID, pero con nombre, contraseña, nivel y correo.
- **`Usuario(Usuario aux)`**: Constructor de copia que crea un nuevo usuario basado en otro.

Métodos Principales:

- **`getId()`**: Obtiene el ID del usuario.
- **`getCorreo()`**: Obtiene el correo del usuario.
- **`getNivel()`**: Obtiene el nivel de acceso del usuario.
- **`getUsuario()`**: Obtiene el nombre de usuario.
- **`getContraseña()`**: Obtiene la contraseña del usuario.
- **`getFotoPerfil()`**: Obtiene la ruta de la foto de perfil del usuario.
- **`setId(int id)`**: Establece el ID del usuario.
- **`setCorreo(String correo)`**: Establece el correo electrónico del usuario.
- **`setNivel(String nivel)`**: Establece el nivel del usuario.
- **`setUsuario(String usuario)`**: Establece el nombre de usuario.
- **`setContraseña(String contraseña)`**: Establece la contraseña del usuario.
- **`setFotoPerfil(String fotoPerfil)`**: Establece la ruta de la foto de perfil.

Operaciones de Gestión de Usuarios:

- **`Usuario_Existe(File archivo, String nombre)`**: Verifica si un usuario con el nombre proporcionado ya existe en el archivo.
- **`listaUsuarios(File archivo)`**: Lee todos los usuarios desde un archivo y los almacena en una lista.
- **`Registrar_Usuario(Usuario act, String direccion)`**: Registra un usuario en un archivo especificado.
- **`guardarUsuarioActual(Usuario act)`**: Guarda la información del usuario actual en un archivo.
- **`obtenerUsuarioActual()`**: Obtiene el usuario actual desde un archivo de texto.
- **`eliminarUsuario(String userHome, Usuario act)`**: Elimina un usuario y sus tareas asociadas desde el sistema.
- **`editarNivel(String userHome, Usuario ant, Usuario act)`**: Modifica el nivel de un usuario en el archivo de usuarios.
- **`editarUsuario(String direcUsuarios, String userHome, Usuario act, Usuario ant)`**: Modifica la información de un usuario en el archivo de usuarios.

Operaciones de Archivos y Directorios:

- **`IDUsuarioNuevo(String direccion)`**: Genera un nuevo ID para un usuario basado en el contenido del archivo.
- **`vaciarArchivo(String direc)`**: Vacía un archivo, eliminando todo su contenido.
- **`eliminarCarpeta(File carpeta)`**: Elimina una carpeta y todos los archivos dentro de ella.

Resumen de Funcionalidades:

- **Gestión de Usuarios:** La clase permite crear, editar, eliminar y buscar usuarios en el sistema, así como verificar si un usuario ya existe en los archivos.
- **Persistencia de Datos:** Los datos del usuario se gestionan a través de archivos, donde se pueden guardar, cargar y modificar según sea necesario.
- **Operaciones con Archivos y Directorios:** Permite la gestión de archivos de usuarios y tareas, incluyendo la creación de nuevos ID, la eliminación de datos y la vaciamiento de archivos.

1.34.Clase: Usuarios

Extiende de: `javax.swing.JFrame`
Descripción:
La clase `Usuarios` representa la ventana principal de la aplicación para gestionar y visualizar los usuarios del sistema. Esta clase proporciona métodos para manipular la lista de usuarios, actualizar la vista y gestionar la interfaz gráfica asociada.

Campos:

- **`lista`**: Lista de objetos `Usuario` que almacena los usuarios existentes en el sistema.
- **`indice`**: Variable que se usa como índice para acceder a los elementos dentro de la lista de usuarios o tareas.
- **`cont`**: Contador entero que se utiliza como índice o para operaciones de control en la lógica de la aplicación.

Constructores:

- **`Usuarios()`**: Constructor que inicializa los componentes gráficos de la interfaz, configura las imágenes usando la clase `Escalar` y carga la lista de usuarios desde un archivo especificado.

Métodos Principales:

- **`main(String[] args)`**: Punto de entrada principal de la ventana. Llama al método para inicializar la interfaz.

- **`setLista(ArrayList<Usuario> aux)`** : Establece la lista de usuarios que se utilizará en la vista y actualiza la tabla según los datos proporcionados. Si no se proporciona un filtro, muestra toda la lista; si se da un filtro, realiza la búsqueda de usuarios correspondientes.
    - **Parámetro:** `aux` – Lista de objetos `Usuario` con la información de los usuarios.
- **`vaciarTabla()`** : Elimina todas las filas de la tabla para que pueda actualizarse con nuevos datos. Se utiliza para limpiar la tabla antes de insertar nuevos registros.

Resumen de Funcionalidades:

- **Gestión de Usuarios:** Permite mostrar y gestionar la lista de usuarios. La vista se actualiza en función de la lista proporcionada, y se puede realizar una búsqueda si se especifica un filtro.
- **Interfaz Gráfica:** La clase hereda de `JFrame`, lo que permite gestionar la interfaz de usuario (ventana) y realizar operaciones gráficas como limpiar y actualizar tablas.

2. Paquete raven_cell:

Este paquete contiene las clases que implementan **3 botones** en las celdas de un `JTable`. Cada botón tiene una acción específica que se ejecuta cuando el usuario interactúa con él dentro de una celda.

2.1.Clase: ActionButton

Extiende de: `javax.swing.JButton`
Descripción:
La clase `ActionButton` extiende `JButton` y crea un botón circular con un efecto visual que cambia de color al presionar el mouse. Este botón tiene un área circular y el color de fondo varía según el estado del mouse (presionado o no). El color se oscurece cuando el botón es presionado, ofreciendo una retroalimentación visual al usuario.

Constructor:

- **`ActionButton()`**
  Inicializa el botón, configurando su área de contenido para que no esté rellena, ajustando los bordes y añadiendo un `MouseListener` para gestionar el cambio de estado visual cuando el mouse se presiona o se libera.

Métodos Principales:

- **`protected void paintComponent(Graphics grphcs)`**
  Redibuja el componente, configurando el área circular y el color de fondo del botón dependiendo del estado del mouse (presionado o no). Si el mouse está presionado, el color de fondo se oscurece para proporcionar una retroalimentación visual.
    - **Parámetro:**
        - `grphcs`: El objeto `Graphics` usado para dibujar el botón.

Resumen de Funcionalidades:

- **Botón Circular con Efecto Visual:**
  El botón tiene una forma circular y cambia de color al ser presionado, proporcionando una respuesta visual intuitiva para el usuario.
- **Interacción con el Usuario:**
  Usa un `MouseListener` para detectar cuando el mouse está sobre el botón y cuando es presionado, alterando su color de fondo en consecuencia.

2.2.Clase: PanelAction

Extiende de: `javax.swing.JPanel`
Descripción:
La clase `PanelAction` es un panel personalizado que contiene botones de acción (como **Editar**, **Eliminar**, y **Ver**) para ser utilizados en una tabla. Cada uno de estos botones tiene asignado un evento para realizar una acción específica sobre una fila de la tabla cuando se presionan. Esta clase gestiona los eventos asociados a los botones y está diseñada para integrarse con una tabla donde cada fila puede tener botones para realizar diversas acciones en esa fila seleccionada.

Constructor:

- **`PanelAction()`**
  Inicializa los componentes del panel, configurando el layout y los elementos visuales, como los botones de acción (Editar, Eliminar, Ver), para que puedan ser utilizados en una tabla.

Métodos Principales:

- **`public void initEvent(TableActionEvent event, int row)`**
  Este método inicializa los eventos para los botones (**Editar**, **Eliminar**, **Ver**) de una fila específica. Asocia los listeners de eventos correspondientes a cada botón para que, cuando el usuario haga clic en uno de ellos, se ejecute la acción correspondiente sobre la fila seleccionada.
  - **Parámetros:**
    - `event`: Un objeto `TableActionEvent` que define las acciones (editar, eliminar, ver) a ejecutar cuando se haga clic en el botón correspondiente.
    - `row`: El índice de la fila de la tabla que será afectada por la acción.

Descripción de la Funcionalidad:

1. **Botones de Acción en una Tabla:**
   El panel contiene botones de acción que se utilizan dentro de una tabla. Estos botones permiten al usuario interactuar con los datos de una fila específica, realizando acciones como editar, eliminar o ver más detalles.
2. **Eventos Asociados a Cada Botón:**
   Los botones (Editar, Eliminar, Ver) tienen eventos (listeners) asociados que permiten realizar las operaciones correspondientes cuando el usuario hace clic en uno de ellos. El evento se dispara para la fila que corresponde a los botones, y las acciones se ejecutan de acuerdo a la selección.
3. **Integración con la Tabla:**
   El panel está diseñado para ser usado en conjunto con una tabla, proporcionando un mecanismo para controlar las acciones de cada fila a través de los botones. El método `initEvent` es responsable de vincular los eventos con las filas de la tabla, de manera que cada botón afecte a la fila correcta.

2.3.Clase: TableActionCellEditor

Extiende de: `javax.swing.DefaultCellEditor`
Descripción:
La clase `TableActionCellEditor` es una clase personalizada que extiende `DefaultCellEditor` para permitir la edición de celdas en una tabla con botones de acción específicos (como **Editar**, **Eliminar** y **Ver**). Esta clase proporciona un editor de celdas que contiene un panel con botones de acción, lo que permite la interacción del usuario con las filas de la tabla. Este editor se utiliza cuando el usuario interactúa con una celda de la tabla para realizar una acción sobre la fila correspondiente.

El editor de celdas reemplaza el editor predeterminado (como `JCheckBox` o `JTextField`) con un panel de botones que permite realizar diferentes operaciones sobre la fila seleccionada, como editar, eliminar o ver más detalles.

Constructor:

- **`TableActionCellEditor(TableActionEvent event)`**
  Este constructor inicializa la clase `TableActionCellEditor`, recibiendo un objeto de tipo `TableActionEvent` que define las acciones a realizar cuando el usuario presiona los botones correspondientes (editar, eliminar o ver). El objeto `event` asocia los métodos que se ejecutarán al hacer clic en los botones del panel.
  - **Parámetros:**
    - `event`: Un objeto `TableActionEvent` que define las acciones (editar, eliminar o ver) a ejecutar cuando el usuario interactúa con los botones de acción.

Métodos Principales:

- **`public Component getTableCellEditorComponent(JTable jtable, Object o, boolean bln, int row, int column)`**
  Este método devuelve el componente que se utilizará para editar la celda de la tabla. En lugar de usar un componente de edición estándar como `JCheckBox` o `JTextField`, este método devuelve un **`PanelAction`**, que es un panel personalizado que contiene los botones de acción (**Editar**, **Eliminar**, **Ver**).
  - **Parámetros:**
    - `jtable`: La tabla en la que se encuentra la celda a editar.
    - `o`: El valor actual de la celda que se está editando.
    - `bln`: Indica si la celda está seleccionada o no.
    - `row`: El índice de la fila de la celda que se está editando.
    - `column`: El índice de la columna de la celda que se está editando.
  - **Retorna:**
    Un componente de tipo **`PanelAction`** que contiene los botones de acción (Editar, Eliminar, Ver) para interactuar con la fila seleccionada.

Métodos Heredados:

- **`cancelCellEditing()`**
  Cancela la edición de la celda. Este método es heredado de `DefaultCellEditor` y puede ser utilizado para revertir cualquier cambio realizado en la celda si el usuario decide no guardar la edición.
- **`getCellEditorValue()`**
  Obtiene el valor que se editará en la celda. Este valor se utiliza cuando se termina la edición de la celda.
- **`isCellEditable()`**
  Verifica si la celda es editable o no, lo que es útil para determinar si se deben habilitar o deshabilitar los botones de acción en el `PanelAction`.

Descripción de la Funcionalidad:

1. **Editor de Celdas con Botones de Acción:**
   La clase reemplaza el editor estándar de celdas en una tabla (como un `JCheckBox` o un `JTextField`) por un `PanelAction`, que contiene botones de acción (**Editar**, **Eliminar**, **Ver**) para interactuar con la fila seleccionada. Este panel ofrece una forma más rica de edición para las celdas que requieren una interacción con botones.
2. **Interacción con la Fila de la Tabla:**
   Cuando el usuario hace clic en una celda que utiliza este editor, el componente de edición reemplaza la celda con el panel de

botones. Los botones permiten ejecutar acciones específicas sobre la fila correspondiente, como editar los datos, eliminar la fila o ver más detalles sobre la entrada seleccionada.

3. **Personalización de las Acciones:**
   El objeto `TableActionEvent` pasado al constructor define las acciones asociadas con los botones de acción. Esto permite una personalización fácil de las operaciones que se deben ejecutar cuando el usuario hace clic en los botones correspondientes.
4. **Facilita la Gestión de Eventos:**
   El editor no solo muestra los botones, sino que también gestiona los eventos que ocurren cuando el usuario interactúa con ellos, proporcionando una manera eficiente de manejar las interacciones con las filas de la tabla.

2.4.Clase: TableActionCellRender

Extiende de: `javax.swing.table.DefaultTableCellRenderer`
Descripción:
`TableActionCellRender` es una clase personalizada que extiende `DefaultTableCellRenderer` y se utiliza para renderizar celdas en una tabla con un panel que contiene botones de acción (como **Editar**, **Eliminar**, **Ver**). Esta clase modifica la forma en que se renderizan las celdas que contienen estos botones, permitiendo personalizar la apariencia de la celda según el estado de selección y la paridad de la fila (si es par o impar).

La principal funcionalidad de esta clase es reemplazar el comportamiento de renderizado estándar para las celdas que contienen botones de acción, ajustando también aspectos visuales como el fondo de la celda, dependiendo de si está seleccionada o no y si la fila es par o impar.

Constructor:

- **`TableActionCellRender()`**
  Constructor predeterminado de la clase `TableActionCellRender`. En este caso, no se especifica ninguna funcionalidad adicional, por lo que simplemente llama al constructor de la clase base (`DefaultTableCellRenderer`).

Métodos Principales:

- **`public Component getTableCellRendererComponent(JTable jtable, Object o, boolean isSelected, boolean bln1, int row, int column)`**
  Este método devuelve el componente que se usará para renderizar la celda en la tabla. En lugar de utilizar el renderizado estándar, este método reemplaza el componente de renderizado predeterminado con un `PanelAction` personalizado que contiene los botones de acción (Editar, Eliminar, Ver). Además, este método personaliza el fondo de la celda según las condiciones especificadas, como si la fila es par o impar y si la celda está seleccionada.
  - **Parámetros:**
    - `jtable`: La tabla en la que se encuentra la celda que se está renderizando.
    - `o`: El valor actual de la celda que se está renderizando (generalmente no se usa directamente en este caso, pero es parte de la API estándar).
    - `isSelected`: Un valor booleano que indica si la celda está seleccionada o no.
    - `bln1`: Un parámetro adicional que indica si la celda tiene el foco.
    - `row`: El índice de la fila de la celda que se está renderizando.
    - `column`: El índice de la columna de la celda que se está renderizando.
  - **Retorna:**
    Un componente `PanelAction` que contiene los botones de acción y tiene el fondo ajustado según la selección y paridad de la fila.

Comportamiento del Renderizado:

1. **Botones de Acción en la Celda:**
   El renderizador reemplaza el comportamiento predeterminado de la celda con un `PanelAction`. Este panel contiene botones de acción como **Editar**, **Eliminar**, y **Ver**. Estos botones permiten al usuario interactuar con las filas de la tabla para realizar operaciones específicas.
2. **Personalización del Fondo:**
   - Si la celda está seleccionada (`isSelected`), el fondo de la celda puede cambiar para indicar su selección.
   - Si la fila es **par** o **impar**, el fondo de la celda puede ajustarse de manera diferente para mejorar la legibilidad (por ejemplo, filas impares pueden tener un fondo de un color, y filas pares otro).
   - Si la celda tiene el foco (`bln1`), se pueden realizar ajustes adicionales en el fondo para resaltar la celda activa.
3. **Aspecto Visual Mejorado:**
   El uso de un panel con botones mejora la interacción visual, haciendo que las celdas no solo contengan datos, sino también controles interactivos. Esta es una técnica común para tablas que necesitan realizar operaciones complejas sobre las filas, como editar o eliminar elementos.

2.5.Interfaz: TableActionEvent

Descripción General:
La interfaz `TableActionEvent` define los métodos necesarios para manejar eventos de acción en una tabla. Estos eventos están relacionados con la interacción del usuario con las filas de la tabla, permitiendo realizar acciones como **editar**, **eliminar** o **ver** los datos de una fila específica. Cualquier clase que implemente esta interfaz debe proporcionar implementaciones para los tres métodos abstractos: `onEdit()`, `onDelete()`, y `onView()`.

Métodos de la Interfaz:

1. **`void onEdit(int row)`**

   Descripción:
   Este método se invoca cuando el usuario intenta editar la fila de la tabla en el índice `row`.
   Parámetros:

   - `row`: El índice de la fila que se está editando. Este índice se utiliza para identificar qué fila debe ser modificada.

   Uso Típico:

   - Abrir un formulario de edición para modificar los datos de la fila.
   - Mostrar un panel de detalles donde se puedan editar los valores de la fila seleccionada.

2. **`void onDelete(int row)`**

   Descripción:
   Este método se invoca cuando el usuario intenta eliminar la fila en el índice `row` de la tabla.
   Parámetros:

   - `row`: El índice de la fila que se está eliminando. Este índice identifica la fila que debe ser eliminada.

   Uso Típico:

   - Eliminar la fila de la tabla y actualizar la vista.
   - Confirmar la acción de eliminación con un cuadro de diálogo antes de proceder.

3. **`void onView(int row)`**

Descripción:
Este método se invoca cuando el usuario desea ver más detalles sobre la fila en el índice `row`.
Parámetros:

- `row`: El índice de la fila que se está visualizando. Este índice permite acceder a los datos específicos de la fila seleccionada.

Uso Típico:

- Mostrar una vista detallada de la fila seleccionada, como un modal o un panel lateral con la información extendida de la fila.

3. Paquete `raven_cell2`:

Este paquete contiene las clases que implementan **3 botones** en las celdas de un `JTable`. Cada botón tiene una acción específica que se ejecuta cuando el usuario interactúa con él dentro de una celda.

3.1.Clase `ActionButton`

La clase `ActionButton` extiende `JButton` y personaliza su apariencia para simular un botón circular. Además, cambia de color cuando el botón es presionado, ofreciendo retroalimentación visual al usuario.

Constructor

- **`ActionButton()`**: Constructor por defecto que configura el botón, eliminando el relleno y el borde. También se añade un *mouse listener* para detectar eventos de presionar y soltar el botón.

Método Principal

- **`paintComponent(Graphics grphcs)`**: Este método sobrescribe el método `paintComponent` de la clase `JComponent` para dibujar un círculo dentro del área del botón. El color del círculo cambia según si el botón está presionado o no.

Herencia y Comportamiento

La clase hereda de `JButton`, por lo que tiene todos los métodos y funcionalidades de un botón estándar de Swing. Los métodos principales heredados incluyen:

- **Acciones y eventos**: `addActionListener()`, `fireActionPerformed()`, etc.
- **Apariencia**: `setIcon()`, `setText()`, `setBorderPainted()`, `setFocusPainted()`, etc.
- **Propiedades de diseño**: `setHorizontalAlignment()`, `setVerticalAlignment()`, etc.

Además, `ActionButton` hereda métodos para manejar eventos de ratón, teclado y otros eventos gráficos, como `repaint()`, `revalidate()`, etc.

Funcionalidad Visual

- El botón simula un círculo, y al presionarlo cambia su color para indicar que está activo.
- No tiene borde ni relleno, lo que contribuye a su apariencia circular.

3.2.Clase `PanelAction`

La clase `PanelAction` extiende `JPanel` y representa un panel de acción en la interfaz de usuario. Este panel contiene botones que permiten realizar acciones sobre una fila de datos, como editar o ver los detalles.

Constructor

- **`PanelAction()`**: Constructor que inicializa los componentes del panel de acción, configurando el panel para su uso dentro de la interfaz de usuario.

Métodos Principales

- **`initEvent(TableActionEvent event, int row)`**: Este método inicializa los eventos de los botones de acción. Asigna los eventos de clic a los botones de edición y visualización. Al hacer clic en estos botones, se ejecuta la acción correspondiente sobre la fila de datos indicada por el índice `row`.

Herencia y Comportamiento

La clase hereda de `JPanel`, lo que le otorga todas las capacidades de un panel estándar de Swing. Algunos métodos heredados clave incluyen:

- **Métodos de interfaz gráfica**: `setLayout()`, `setBackground()`, `setBorder()`, `setPreferredSize()`, etc.
- **Métodos de eventos**: Métodos como `addMouseListener()`, `repaint()`, `requestFocus()`, entre otros, permiten gestionar la interacción con el usuario y la actualización visual del panel.

Funcionalidad

- El panel es utilizado para contener y gestionar botones de acción que permiten interactuar con una fila de datos, generalmente en una tabla.
- El método `initEvent` configura la acción de los botones para que realicen las operaciones correspondientes sobre una fila de datos, como editar o ver detalles.

3.3.Clase `TableActionCellEditor`

La clase `TableActionCellEditor` extiende `DefaultCellEditor` y proporciona un editor personalizado para las celdas de una tabla. Este editor muestra un panel con botones de acción (como "editar" y "ver") dentro de una celda de la tabla.

Constructor

- **`TableActionCellEditor(TableActionEvent event)`**: Constructor que inicializa el editor de celda con un evento de acción. Este evento se maneja cuando el usuario interactúa con los botones dentro de la celda.

Métodos Principales

- **`getTableCellEditorComponent(JTable jtable, Object o, boolean bln, int row, int column)`**: Sobrescribe el método `getTableCellEditorComponent` de `DefaultCellEditor` para proporcionar un componente personalizado. En lugar de usar un componente de celda estándar, se utiliza un `PanelAction` que contiene los

botones de acción dentro de la celda.
Parámetros:
- `jtable`: La tabla donde se edita la celda.
- `o`: El valor de la celda (no se usa en este caso).
- `bln`: Indica si la celda está seleccionada (no se usa en este caso).
- `row`: Índice de la fila de la tabla donde se encuentra la celda.
- `column`: Índice de la columna de la tabla donde se encuentra la celda.

- **Retorno**: Devuelve el componente que se mostrará en la celda, en este caso, un `PanelAction` con los botones de acción.

Herencia y Comportamiento

La clase hereda de `DefaultCellEditor`, por lo que tiene las funcionalidades estándar de un editor de celdas en una tabla. Los métodos heredados permiten gestionar el proceso de edición de la celda, como:

- **Manejo de eventos**: `addCellEditorListener()`, `removeCellEditorListener()`, etc.
- **Control de edición**: `cancelCellEditing()`, `stopCellEditing()`, `getCellEditorValue()`, etc.

Funcionalidad

- Este editor está diseñado para ser utilizado en celdas de tablas que requieren botones de acción dentro de cada celda, como botones de "editar" o "ver".
- En lugar de mostrar el valor simple de la celda, el editor proporciona una interfaz más rica (un `PanelAction` con botones).

3.4.Clase `TableActionCellRender`

La clase `TableActionCellRender` extiende `DefaultTableCellRenderer` y proporciona un renderizador personalizado para las celdas de acción dentro de una tabla. Este renderizador se utiliza para mostrar un panel con botones de acción (como editar, ver, etc.) dentro de una celda de la tabla.

Constructor

- **`TableActionCellRender()`**:
  Constructor predeterminado de la clase. No realiza ninguna acción adicional pero es necesario para crear una instancia de la clase.

Métodos Principales

- **`getTableCellRendererComponent(JTable jtable, Object o, boolean isSeleted, boolean bln1, int row, int column)`**:
  Sobrescribe el método `getTableCellRendererComponent` de `DefaultTableCellRenderer` para proporcionar un componente personalizado (un `PanelAction`) en lugar de un simple componente de celda.
  Parámetros:
  - `jtable`: La tabla en la que se está renderizando la celda.
  - `o`: El valor de la celda (este parámetro no se utiliza en este caso).
  - `isSeleted`: Indica si la celda está seleccionada.
  - `bln1`: Parámetro no utilizado.
  - `row`: El índice de la fila de la tabla donde se encuentra la celda.
  - `column`: El índice de la columna de la tabla donde se encuentra la celda.
- **Retorno**:
  Devuelve el componente que se mostrará en la celda, que es un `PanelAction` que contiene los botones de acción.

Herencia y Comportamiento

- La clase hereda de `DefaultTableCellRenderer`, lo que le permite renderizar celdas dentro de una tabla.
- La clase implementa la interfaz `TableCellRenderer`, que es utilizada por las tablas (`JTable`) para renderizar las celdas.

Funcionalidad

- El renderizador está diseñado para las celdas que contienen acciones (como "editar", "ver", etc.). En lugar de mostrar solo texto o valores, el renderizador muestra un `PanelAction`, que es un panel con botones de acción.

3.5.Interfaz `TableActionEvent`

Esta interfaz define eventos de acción para las celdas de una tabla. Proporciona métodos que permiten ejecutar acciones de edición y visualización en las filas de la tabla. Las clases que deseen manejar estas acciones pueden implementar esta interfaz.

Métodos

1. **`onEdit(int row)`**
   Acción para editar una fila de la tabla.
   - **Parámetros:** `row` - El índice de la fila que se va a editar.
2. **`onView(int row)`**
   Acción para ver los detalles de una fila de la tabla.
   - **Parámetros:** `row` - El índice de la fila que se va a visualizar.