****

**Universidad Nacional de General Sarmiento**

**Programación III**

**Trabajo Práctico:**

**“El equipo ideal”**

**Profesores:**

* Patricia Bagnes.
* Javier Marenco.

**Grupo:**

* Andrés Geiser
  + 2018/42239150
  + geiserandres00@gmail.com
* Hernán Rodrigo Fernández
  + 2017/41584220
  + fernandezhr\_1998@hotmail.com

**Objetivo**

El objetivo que planteamos resolver en este trabajo práctico consiste en implementar una aplicación que muestre la conformación optima de un equipo de desarrollo a partir de una cierta cantidad de empleados con distintos puestos prefijados por el usuario y los respectivos lazos de convivencia entre ellos, lo cual determinara la posibilidad de generar un equipo optimo con las condiciones dadas.

**Responsabilidad de clases**

El proyecto cuenta con 6 paquetes en los que están distribuidas las clases según su funcionalidad:

* **Archivos**: se encuentran los archivos que contienen una serie de nombres y apellidos que se utilizan para hacer una combinación entre ellos y posteriormente generar empleados con el nombre de dicha combinación.
* **Controlador**: clases encargadas de comunicar los distintos componentes visuales con aquellas que implementan la lógica interna. Donde se define el comportamiento de los componentes de la interfaz.
* **Iconos**: donde se encuentran los archivos que se utilizan como iconos en los componentes de la interfaz visual.
* **Modelo**: se encuentran las clases encargadas de poner en funcionamiento la lógica interna del programa y la suite de Test correspondiente a las mismas.
* **Principal**: contiene a la clase principal que se encarga de inicializar las clases visuales y desencadenar el funcionamiento del programa.
* **Vista**: donde se encuentran las clases visuales, que contienen el código que implementa la interfaz correspondiente al MainFrame y a los distintos paneles.

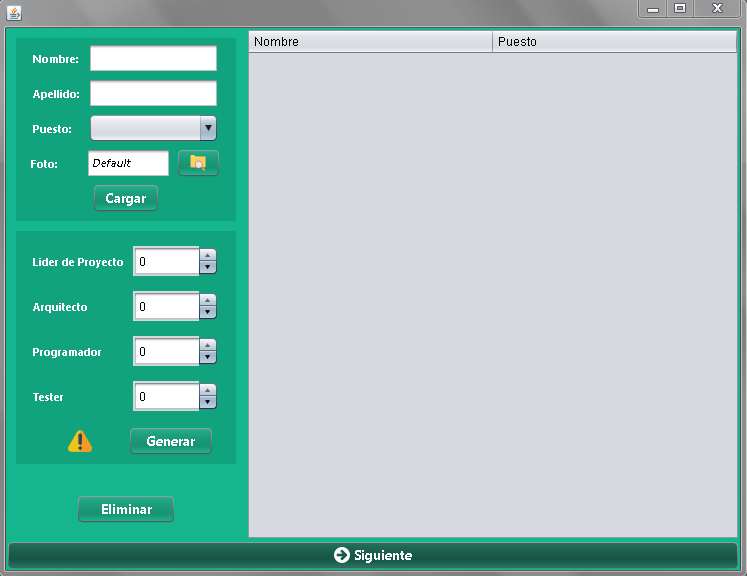
A continuación, presentaremos a cada una de las clases de los paquetes más importantes, describiendo sus variables internas, sus métodos y su rol general dentro del programa.

PAQUETE

**VISTA**

**Clase VentanaPrincipal**

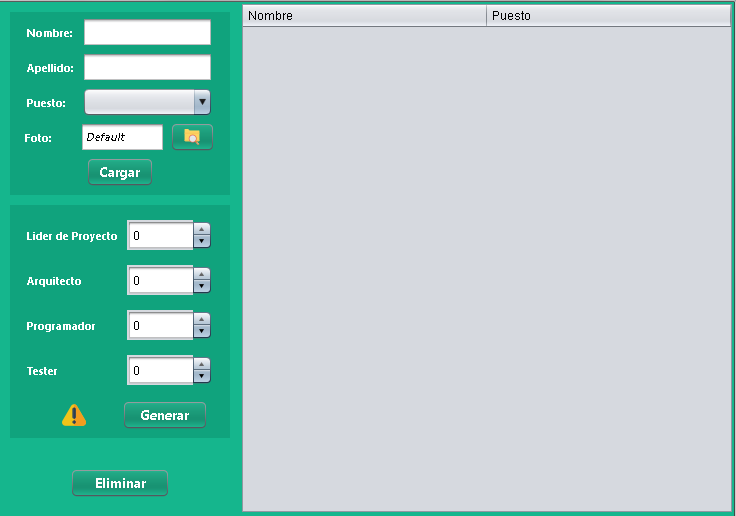
Implementa la interfaz principal de la aplicación.



La ventana solo cuenta con un panel central (el cual cambiara en cuatro diferentes) y un botón en la parte inferior con el cual podremos cambiar el panel central para pasar hacia otras funcionalidades de la aplicación.

**Clase CargarEmpleado**

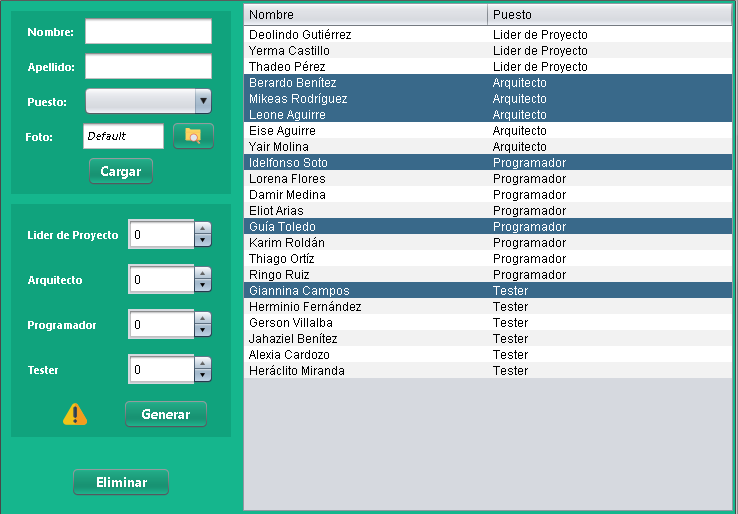
Implementa uno de los cuatro paneles centrales de la ventana principal. Tiene el siguiente aspecto:



Es el primer panel que se visualiza y el cual usamos para cargar empleados y mostrarlos en pantalla.

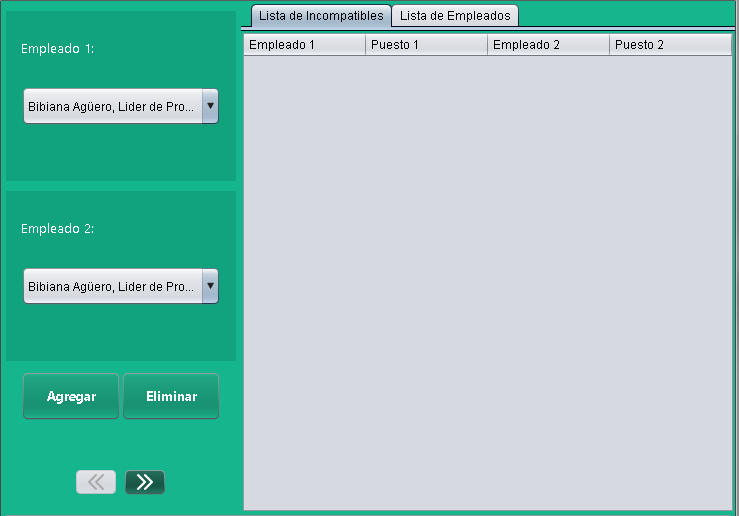
Tiene un panel a la derecha con una tabla que mostrara la lista con los empleados que se irán cargando y a la izquieda otro panel con otros dos paneles más pequeños. El primero contiene dos JTextField para escribir el nombre del empleado, un JComboBox con el cual elegiremos el puesto del empleado, un botón que abrirá un selector de archivos para poder subir una foto al empleado y por último un botón “Cargar” que tomará esos datos y los colocará en la tabla. El segundo cuenta con cuatro JSpinner con los que podremos seleccionar una determinada cantidad de empleados y generarlos con un nombre que la aplicación le asignara, también tiene en la parte inferior izquierda un icono el cual al pasar el mouse por encima dice la cantidad de empleados que la aplicación puede generar.

Por último, tiene un botón con el cual podremos eliminar los empleados. Esto se hace seleccionándolos en la tabla y luego presionando el botón “Eliminar”.



**Clase CargarDatos**

Implementa uno de los cuatro paneles centrales de la ventana principal. Tiene los siguientes aspectos:



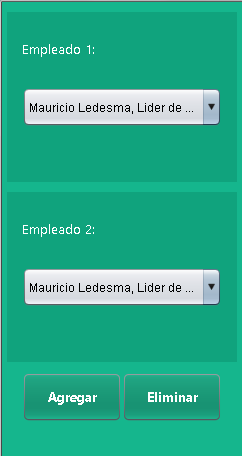


Es el panel que nos permitirá cargar la lista de empleados incompatibles y los requerimientos específicos del equipo.

Tiene un panel a la derecha que contiene la tabla de empleados incompatibles entre sí y otra tabla con los empleados. En la parte izquierda hay otro panel que cuenta con un panel central (que será el **CargarIncompatibles** o **CargarCotas**) el cual podremos cambiar presionando los botones que se encuentran abajo.

**Clase CargarIncompatibles**

Implementa el siguiente panel:

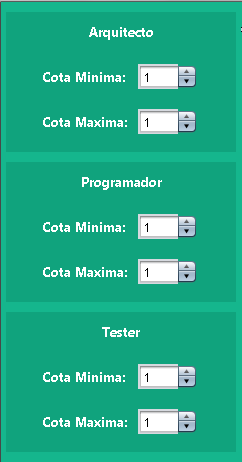


Este panel nos permitirá elegir dos de los empleados cargados mediante unos JComboBox y agregarlos a la tabla de empleados incompatibles (siempre y cuando hallamos elegido dos empleados distintos), también tiene un botón “Eliminar” con el que podremos borrar de la tabla la relación.

****

**Clase CargarCotas**

Implementa el siguiente panel:

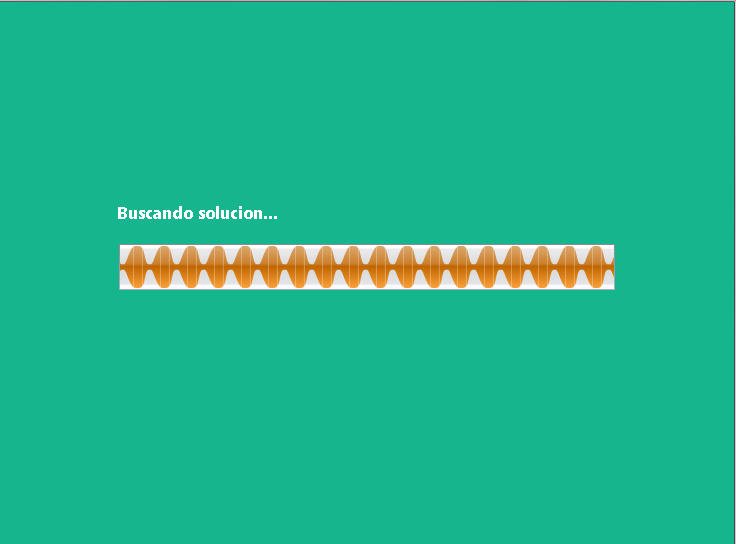


Este panel cumplirá la función de permitirnos elegir los requerimientos que deseamos para el equipo, es decir, una cota mínima/máxima para los arquitectos, programadores y testers.

Tiene dos JSpinner para elegir el mínimo y el máximo para cada puesto.

**Clase BuscarSolucion**

Implementa el panel que se visualizara mientras se ejecuta el algoritmo que buscara el conjunto de empleados que cumpla con las condiciones:



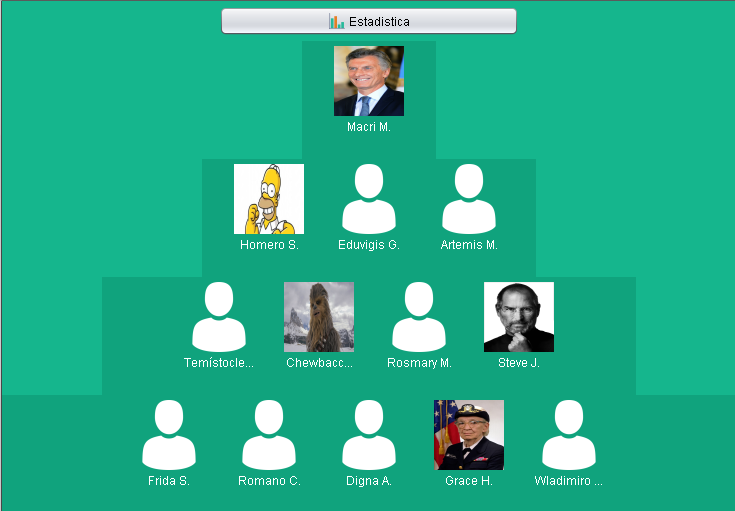
Su única funcionalidad es informar que se está buscando la solución a la problemática de este trabajo practico.

Los únicos componentes que contiene es un JLabel y un JProgressBar indeterminado.

**Clase MostrarSolucion**

Implementa el ultimo panel que se visualizara en la ventana principal. Puede tener 2 aspectos distintos:

Si se encontró un conjunto que cumpla las condiciones se mostrara de la siguiente forma:



Tiene un botón “Estadística” en la parte superior el cual mostrara el tiempo que ha tardado el algoritmo en encontrar la solución y la cantidad de veces que entro en el caso base.

Luego tiene una serie de 4 paneles uno abajo del otro en los cuales se visualizarán los empleados elegidos. El primero contendrá al líder de proyecto, el segundo a los arquitectos, el tercero a los programadores y el ultimo a los testers. Cada empleado se visualizará con su respectiva foto.

En caso de que no se haya encontrado una solución mostrara el siguiente panel:



PAQUETE

**MODELO**

**Clase Empleado**

Esta clase modelara a un empleado el cual tiene un nombre y un puesto dentro del equipo.

**Variables de clase**

* *String nombre*: representa el nombre del empleado.
* *String puesto*: representa el puesto del empleado.

Luego la clase tiene métodos getNombre() y getPuesto() que son llamados por otras clases, también posee un método equals para verificar que dos empleados sean distintos.

**Clase Requerimiento**

Esta clase será utilizada para guardar los requerimientos por puestos establecidos por el usuario, solo posee métodos de set() y get() para sus variables.

**Variables de clase**

* *int minLideres,maxLideres* : representa la cantidad mínima y máxima de líderes de proyecto que debe tener el equipo. Por defecto es inicializada en 1.
* *int minArquitectos, maxArquitectos*: representa la cantidad mínima y máxima de arquitectos que debe tener el equipo. Por defecto es inicializada en 1.
* *int minProgramadores, maxProgramadores*: representa la cantidad mínima y máxima de programadores que debe tener el equipo. Por defecto es inicializada en 1.
* *int minTesters, maxTesters*: representa la cantidad mínima y máxima de testers que debe tener el equipo. Por defecto es inicializada en 1.

**Clase Instancia**

Esta clase representa un estado actual del problema a resolver, contiene la lista de empleados, las relaciones entre ellos y los requerimientos de los puestos especificados.

**Variables de clase**

* *Grafo malasRelaciones:* contiene las relaciones entre los empleados.
* *ArrayList<Empleados> empleados*: contiene a todos los empleados generados por el usuario.
* *Requerimiento requerimiento*: contiene las especificaciones por puesto que fija el usuario.

Luego la clase contiene un método sonIncompatibles(int i, int j) que es utilizado para verificar la relación entre dos empleados de manera auxiliar y los demás métodos retornan la lista de empleados y las cotas especificadas en requerimientos.

**Clase Assert, testModelo y testSolverEquipo**

Estas clases contienen la suite de test que controlan los métodos de modelo y solver que contienen 24 y 11 test respectivamente, luego la clase Assert es utilizada para probar si dos listas de empleados son iguales, es decir que representan el mismo equipo solución.

**Clase grafo**

Esta clase será utilizada para modelar las distintas relaciones con la estructura de grafo, donde una buena relación significará que existe una arista entre ambos vértices y de lo contrario habrá mala relación.

**Variables de clase**

* *boolean [][] A*: representa la matriz de adyacencia interna del grafo.

**Métodos de clase**

* *void agregarArista(int i, int j)* : se encarga de agregar una arista entre los vértices **i** y **j** colocando en esa posición de la matriz true si existe arista (para nuestro problema true si tienen buena relación laboral).
* *void borrarArista(int i, int j)* : se encarga de eliminar la arista entre los vértices **i** y **j** volviendo false la posición correspondiente de la matriz **A**.
* *boolean existeArista(int i, int j):* verifica si existe una arista entre los vértices **i** y **j**. Es utilizado para verificar la relación entre dos empleados.
* *set<Integer> vecinos(int i):* retorna una colección con aquellos vértices tengan relación con el vértice **i**.

**Clase Modelo**

Esta clase contiene la lógica interna principal del programa, organiza la carga de empleados, las relaciones entre los empleados y gestiona la solución del equipo de desarrollo final.

**Variables de clase**

* *ArrayList<Empleado> empleado*: contiene la lista de todos los empleados que se generaron.
* *Grafo malasRelaciones*: representa a los vértices como empleados y a las aristas como buenas relaciones entre sí.
* *Requerimiento requerimiento*: contiene los valores máximos y mínimos elegidos para cada puesto dentro del equipo.
* *ArrayList<Empleado> solución*: contiene a los empleados validos que cumplen los requisitos y que son parte del equipo final.
* *Solver solver*: referencia a la clase que se utilizara para modelar el equipo óptimo.

**Métodos de clase**

* *void agregarEmpleado(String nombre, String puesto):* si el empleado es válido y si la lista aun no fue confirmada lo agrega a *empleados*.
* *void eliminarEmpleado(String nombre):* si el empleado es válido y si la lista aun no fue confirmada lo elimina de *empleados.*
* *void confirmarListaDeEmpleados()*: si los empleados actuales de *empleados* cumplen con los requisitos mínimos inicializa *malasRelaciones* con el correspondiente tamaño.
* *void agregarMalaRelacion(String nombreE1, String nombreE2) y eliminarMalaRelacion*: si la lista fue confirmada y los dos empleados existen agrega (buena relación) o elimina(mala relación) un la arista entre los dos empleados.
* *void resolver():* inicializa a *solver* y llama a .resolver con una nueva instancia armada con la lista de empleados, malasRelaciones y los requerimientos por puestos, almacenando en *solucion* a los empleados que integran el equipo final*.*
* *String getEstadisticas()*: devuelve una cadena con las estadísticas finales de la ejecución del solver

**Clase Solver**

Esta clase implementa un algoritmo de fuerza bruta basado en clique máxima, pero con adiciones propias de nuestro problema. Dado un grafo g encuentra la clique que contiene el grupo mas grande de empleados que cumplen los requisitos.

**Variables de clase**

* *Instancia instancia*: representa el conjunto de datos que recibe la clase para buscar el equipo óptimo.
* *Set<Integer> conjuntoActual:* contendrá los índices de los empleados que se formaran para conseguir el equipo óptimo.
* *ArrayList<Empleado>:*contendrá los empleados que pertenezcan a la solución final, es decir al equipo óptimo con las condiciones dadas.
* *Int totalIteraciones y double Tiempototal:*representan variables de estadística para controlar el tiempo total de ejecución y la cantidad de iteraciones del caso base.

**Métodos de clase**

* *ArrayList<Empleado> resolver():* inicializa a los conjuntos, llama al método encargado de la recursión con el elemento 0 y devuelve el equipo óptimo formado con los empleados de la *instancia*.
* *void recursión(int inicial):* implementa el algoritmo general recursivo de fuerza bruta, se lo llama inicialmente con algún índice desde el que comenzara a armar todos los equipos posibles. Cuando el inicial sea igual al tamaño del equipo de la instancia, es decir la cantidad de empleados sea igual al inicial, se comprueba que entre los empleados del equipo formado actualmente no existan incompatibilidades y que se cumplan con los requisitos preestablecidos de los distintos puestos.

**(Creo que haría falta tal vez dibujar un esquema para verlo más fácil)**

Luego se forma el equipo máximo que cumple los requisitos, de contener mas empleados que la solución optima esta pasa a ser la mejor solución encontrada. De esta manera se lanzará la recursión con todos los iniciales posibles para lograr encontrar el equipo optimo que cumpla con los requisitos. La cantidad de veces que se ejecutara el caso base corresponde a 2*n* con *n* igual a la cantidad de empleados del conjunto inicial.

**(Un poco incompleto! ¿Deberíamos poner una foto mostrando la complejidad del caso base?)**

PAQUETE

**CONTROLADOR**

**Clase ListaDeNombres**

Clase que implementa un conjunto de nombres (nombre + apellido) que se obtienen a partir de los archivos que se encuentran en el paquete **archivos**. Fue creada con el objetivo de poder crear empleados dinámicamente.

**Variables de clase**

* *ArrayList<String> listaDeNombres:* contendrá las combinaciones de nombres y apellidos.

**Métodos de clase**

* *void generarNombres():* pone en marcha la lectura de los archivos “nombres.txt” y “apellidos.txt”. Luego por cada nombre obtenido lo combina con cada uno de los apellidos.
* *List<String> leerFichero(String fichero):* devuelve una lista de string que se obtienen al leer cada una de las líneas del archivo pasado como parámetro.
* *String dameUno():*retorna un string aleatorio de la lista de nombres y lo elimina de la misma.
* *void eliminar(String n):* elimina de la lista de nombres el string pasado como parámetro.
* *int cantidad():* retorna la cantidad de nombres de contiene la lista.

**Clase FotosDeEmpleados**

Clase en la cual guardaremos las fotos de los empleados que se irán cargando mediante variables y métodos estáticos.

**Variables de clase**

* *static HashMap<String, ImageIcon> fotosDeEmpleados:* almacenará las fotos de los empleados, donde la clave será el nombre del empleado.

**Métodos de clase**

* *static agregarFoto(String nombre, ImageIcon foto):* agrega la imagen que recibe como parámetro usando el String ‘nombre’ como clave.
* *static eliminar(String nombre):* elimina la foto correspondiente a la clave que se recibe como parámetro.
* *static ImageIcon getFoto(String nombre):*retorna la imagen asociada a la clave que se recibe como parámetro.

**Clase CtrlVentanaPrincipal**

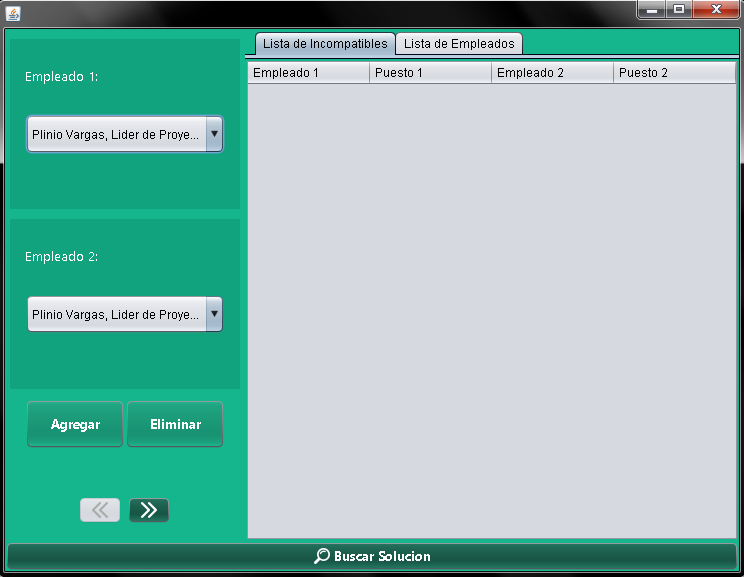
Esta clase se encarga de controlar el comportamiento de la interfaz principal y lógica de la aplicación haciendo uso también de los controladores de los distintos paneles.

De esta clase explicaremos el método principal que es el **cambiarDePanel()** al cual se lo llama al presionar el botón de la parte inferior de la aplicación. Lo que hace este método básicamente es cambiar el panel central de la ventana principal dependiendo de que panel central se encuentre actualmente e iniciar su respectivo controlador. Por ejemplo:



Presionamos botón “Siguiente”

Ventana principal con el panel CargarEmpleados como panel central.



Ventana principal con el panel CargarDatos como panel central.

**Clase CtrlCargarEmpleados**

Clase encargada de manejar la información que se obtiene en el panel CargarEmpleados que mencionamos anteriormente y pasársela a la lógica de la aplicación.

**Variables de clase**

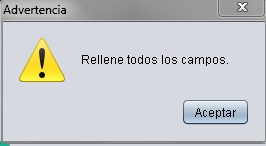
* *Modelo modelo:* lógica de la aplicación.
* *CargarEmpleado panelCargarEmpleado:*panel con el cual esta clase interactuara.
* *ListaDeNombres listaDeNombres:*lista la cual usaremos para generar empleados dinámicamente.

**Métodos de clase (más relevantes)**

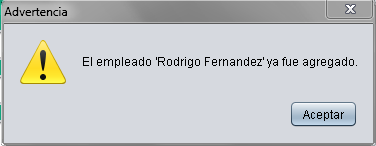
* *void elegirFoto():* abre un selector de archivos filtrando solo los archivos de tipo .png o .jpg.
* *void cargarEmpleado()*: obtiene el nombre y el apellido que se escribió en los JTextField, el puesto del JComboBox y se los pasa a la lógica. Además, guarda la foto del empleado en caso de que haya elegido alguna, sino se le asigna una imagen por defecto.
* *void generarEmpleados():* crea dinámicamente empleados dependiendo de la cantidad que halla en los JSpinner, asignándole un nombre el cual se obtiene de la listaDeNombres.
* *void eliminarEmpleados():* elimina tanto de la lógica como de la tabla los empleados seleccionados.
* *void actualizarVista():* actualiza la visualización de los distintos componentes del panel.
* *boolean esPosibleCargarEmpleado():* devuelve verdadero si los campos de información están llenos y el nombre no está repetido.
* *boolean sePuedeGenerar:* retorna verdadero si la cantidad de empleados que se desea crear no supera la cantidad de nombres que tiene la lista.
* *boolean tieneSuficientesDatos:* retorna verdadero si al menos se ha cargado un empleado de cada puesto.

Esta clase lanza ventanas emergentes como las siguientes:

En caso de que se intente cargar un empleado sin escribir el nombre o apellido o sin haber especificado un puesto.



En caso de que se intente cargar un empleado con nombre igual a uno que ya está cargado.

****

**Clase CtrlCargarDatos**

Esta clase se encarga de controlar el comportamiento del panel CargarDatos y lógica de la aplicación haciendo uso también de los controladores de los paneles CargarIncompatibles y CargarCotas.

Esta clase no hace mucho contacto directo con la lógica a diferencias de las demás.

El único método destacable de esta clase es el **actionPerformed()** el cual hará cambiar el panel izquierdo mediante los botones.

**Clase CtrlCargarIncompatibles**

Clase que se encarga de controlar el comportamiento del panel CargarIncompatibles y lógica.

**Variables de clase**

* *Modelo modelo: lógica de la aplicación.*
* *CargarIncompatibles panelCargarIncompatiles:* panel con el cual esta clase interactuara.
* *JTable tablaIncompatbiles:*tabla que esta clase usara para representar los empleados incompatibles.

**Métodos de clase**

* *void agregarIncompatibles():* toma los dos empleados seleccionados de los JComboBox y se los pasa a lógica para agregar la mala relación entre ellos.
* *void eliminaIncompatibles():* elimina la mala relación de los empleados que selecciono en la tabla.
* *void actualizarTablaDeIncompatibles():*actualiza la tabla agregando una fila mas con los datos de los dos JComboBox.

**Clase CtrlCargarCotas**

Esta clase se encarga de manejar la información que proveerá el panel CargarCotas y hacer la comunicación respectiva con la lógica.

**Variables de clase**

* *Modelo modelo: logica de la aplicación.*
* *CargarCotas panelCargarCotas:* panel con el cual esta clase interactuara.

**Métodos de clase**

* *void ejecutar():*toma los valores actuales delos JSpinner que marcan las cotas mínimas/máximas de los diferentes puesto y se los pasa a la lógica.
* *boolean tieneCotasValidas():* retorna verdadero si algunos de los valores de las cotas mínimas supera a la máxima(es decir, si la cota mínima de programadores supera a la cota máxima de programadores, etc.).

**Clase CtrlMostrarSolucion**

Clase encargada de mostrar visualmente en el panel MostrarSolucion el conjunto de empleados que la lógica encontró.

**Variables de clase**

* *Modelo modelo: lógica de la aplicación.*
* *MostrarSolucion panelMostrarSolucion:* panel con el cual esta clase interactuara.

**Métodos de clase**

* *void cargarEquipoElegido:* toma la solución de la logica y obtiene el nombre de cada empleado para buscar su foto en la clase FotosDeEmpleado y agregarla al panel correspondiente.