Informe de Documentación del

Trabajo Práctico N°3: El equipo ideal

Comisión: 02 (Mañana)

Alumnos:

-Trejo Fernando (43.986.607)

-García Matías (41.385.966)

-Gordillo Fabricio (42.948.240)

Objetivos:

Se nos encargó realizar una aplicación con la cual el usuario pueda armar un equipo de trabajo compuesto por personas con diferentes roles y encontrar un conjunto lo más calificado posible de personas que cumpla estos requerimientos, y que no contenga ningún par de personas incompatibles.

Dificultades:

Cuando entregamos el TP sabíamos que no estaba funcionando el solver y SwingWorker. A partir de esto y los comentarios que nos hizo Daniel, volvimos a reveer el código y ver que pudimos haber hecho mal o quizás deberíamos mejorar.

Una de las cosas que nos dijo Daniel era que había que especificarle un poco mejor al usuario, los datos que debía ingresar, así que modificamos y agregamos label para

informativos para el usuario.

También agregamos en la interfaz que el usuario pueda agregar las incompatibilidades que desee.

Además pudimos solucionar el tema del Thread gracias a que Daniel nos dijo que no lo estábamos iniciando, luego por otro lado el tema del Solver lo pudimos solucionar debuggeando y haciendo casos de prueba. El problema estaba en la solución que no estaba comparando.

Implementación:

Clase Main: Clase de la interfaz gráfica de la aplicación. Atributos:

- JFrame frame: Objeto de la interfaz.
- JTextField txtNombre: Objeto de la interfaz
- JTextField txtRating: Objeto de la interfaz.
- JComboBox<String> cbRoles :Objeto de la interfaz
- JComboBox<String> cbIncompatibilidad1: Objeto de la interfaz
- JComboBox<String> cbIncompatibilidad2:Objeto de la interfaz
- JButton btnAgregarPersona :Objeto de la interfaz
- JButton btnAgregarRol :Objeto de la interfaz
- JTextField txtNuevoRol:Objeto de la interfaz
- ManejoEquipo manejoEquipo: Determina un .
- Font fuente : Determina una fuente.
- Color colorFondo: Determina el color del fondo.
- Color colorBotones: Determina el color del objeto botón.
- Color colorBlanco: Deja guardado una variante de color blanco.
- JTextField textFieldCantidad: Objeto de la interfaz

- JTextArea txtPersonas: Objeto de la interfaz

- Simulacion simulación: Determina una Simulación.

- JProgressBar progressBar: Objeto de la interfaz

Métodos:

void limpiarFrame() : Remueve los elementos y el layout del frame

void actualizarFrame() : Actualiza los elementos del frame

void inicializarPanelAgregarRol() : Crea el panel de la primera pantalla para agregar los roles que se necesitan.

void inicializarPanelIncompatibilidades() : Crea el panel para poder cargar las incompatibilidades, además tiene un botón para poder mostrar el equipo final.

void inicializarPanelMostrarSolucion(): Crea el panel con la solución

void inicializarPanelAgregarPersona(): Crea el panel para agregar las personas

void cargarSolucion(): Utilizamos GridBagLayout para poder posicionar los elementos y luego mostramos el equipo completo, la mejor solución y las incompatibilidades.

void agregarPersona(): Agrega persona a arraylist del manejo de equipo y limpia los txt.

void agregarRol():Agrega rol a la lista roles y limpia los txt.

void agregarIncompatibilidad():Agrega la incompatibilidad y limpia los txt.

Clase Solver: Clase con el algoritmo que resuelve

Atributos:

Map<String, Integer> contadorPersonasEnRol(): Objeto Map que contiene personas

int bestRating = 0: Se usa para ir iterando sobre los ratings

List<Persona> solución(): Lista que contiene las personas agregadas a

la solución definitiva.

List<Persona> solucionActual; //Lista nueva para la solución actual que va cambiando a medida que hace comprobaciones

Métodos:

boolean solve(int indicePersona): Metodo recursivo que hace la funcion de backtracking

boolean esCompatible(Persona persona): Verifica que una persona sea compatible con las demás

boolean estanTodosLosRolesRepresentados(): Verifica que los roles esten completos para la solucion

void agregarPersonaAEquipo(Persona personaActual): Agrega a la persona actual a solucionActual, además de eso suma uno en contadorPersonasEnRol para ir contando cuantas personas hay en cada rol.

removerPersonaDeEquipo(Persona personaActual): Elimina a la persona actual de solucionActual, además de eso resta uno contadorPersonasEnRol para ir contando cuantas personas hay en cada rol.

List<Persona> getSolucion():Contiene la solucion con las personas ideales

Clase Simulación: Clase que tiene el swingWorker

Atributos:

JTextField resultJTextField: Objeto de la interfaz

JProgressBar barraProgreso: Objeto de la interfaz para la barra

ManejoEquipo manejoEquipo: Determina un manejoEquipo.

Color colorBotones: Determina un color.

Font fuente: Determina una fuente.

Métodos:

void doInBackground(): Realiza el solver y devuelve la solución.

void done(): Verifica si finalizó el solver o si ocurre algún problema, devuelve una lista de personas en un grid de forma vertical para que quede una encima de la otra.

Clase ManejoEquipo: Clase que contiene el manejo del

equipo Atributos:

List<Persona> personas: Lista que contiene personas

List<Rol> roles: Lista que contiene roles

List<Incompatibilidad> incompatibilities: Lista que contiene

Métodos:

void agregarPersona(Persona person): Agrega personas a la lista

personas void agregarRol(Rol role) Agrega roles a la lista roles

void agregarIncompatibilidad(Incompatibilidad incompatibility):
Agrega incompatibilidades al array Incompatibilities

void mostrarPersonas() Ejecuta un print de las personas con sus roles y sus rating

void mostrarRoles() Ejecuta un print con los roles y la cantidad que se necesitan

void mostrarIncompatibilidades()Ejecuta un print con las incompatibilidades

List<Persona> getPersonas() Get de la lista personas

List<Rol> getRoles() Get de la lista roles

List<Incompatibilidad> getIncompatibilidades() Get de la lista incompatibilidades

Clase Incompatibilidad: Clase de la incompatibilidad que asocia 2 personas

Atributos:

Persona persona1: Clase Persona

Persona persona2: Clase Persona

Métodos:

Incompatibilidad(Persona persona1, Persona persona2) //Constructor

Clase Persona: Crea una persona

Atributos:

String nombre: nombre que va a tener la persona

String rol: rol de la persona

Int rating: rating de la persona

Métodos:

Persona(String nombre, String rol, int rating)//Constructor

boolean equals(Object obj)// Verifica que 2 clases sean iguales

Clase Rol:
Atributos:

String nombre: Nombre del rol

int cantidad: Cantidad de personas necesarias para el rol