Universidad Nacional de General Sarmiento

Licenciatura en Sistemas
Tecnicatura Superior en Informática

Trabajo Práctico n°3
La hora, referí!

Programación 3

segundo semestre 2021

Comisión 1, turno noche.

Profesores: Javier Marenco
Patricia Bagnes

Integrantes: Gustavo Córdoba cordobagustavoj@gmail.com
Julieta Rocha julietamd45@gmail.com

1. Introducción

En el siguiente trabajo práctico se busca implementar un algoritmo goloso,para asignar árbitros a partidos, simulando un campeonato de fútbol.

2. Desarrollo

En este trabajo se utiliza la estructura MVC la cual separa el código en:

● ***logica(controlador)***: *es la parte lógica del código que se encarga de hacer funcionar el algoritmo goloso y actualizar la vista. Tambien se utilizo un paquete para los test que verifican el correcto funcionamiento de la logica*

● ***Vista*** *: recibe los cambios realizados por la parte lógica y genera en pantalla dichos cambios de forma gráfica.*satisfactoriamente.

3. Complicaciones y soluciones:

Al finalizar con las pautas del enunciado, nos planteamos que el algoritmo goloso solo da una única solución, así que pensamos en una forma de generar una asignación aleatoria en donde se mantenga el mismo promedio. Para este problema, en un principio, pensamos que el algoritmo goloso tome la decisión de elegir un árbitro aleatorio (siempre y cuando estos tengan el mismo promedio en el partido) luego de pensar esto nos dimos cuenta de que esto rompería el test de asignar arbitraje y tendríamos que adaptar el test a la nueva implementación del algoritmo. Pero, esto no nos pareció lo correcto, con esto en cuenta notamos que si intercambiamos dos árbitros, el promedio se mantiene igual para ambos. Entonces, creamos un método para intercambiar dos árbitros, pero presentamos otro problema que se relaciona con los partidos, estos partidos tenían los nombres de los árbitros originales, al hacer el intercambio esto debería cambiar, así que con un Map<Árbitro, ArrayList > solucionamos este problema y además obtuvimos un método para cambiarle el nombre a un árbitro.

En relación a cargar los archivos JSON, decidimos crear un paquete donde esten las clases encargadas de cargar los archivos. Estos archivos JSON guardan un ArrayList (para el calendario) y ArrayList (para la lista de equipos). Estas ArrayList son las que recibe la clase Torneo

Otro inconvenienteque tuvimos fue con las vistas. Decidimos que cada fecha del calendario sea un panel, este panel es la clase FechaPanel, donde se mostraba una tabla el problema que surgió fue que si hay muchos partidos, se generaba una tabla con todos los partidos pero no se mostraban todos por el tamaño del panel. Investigando un poco encontramos que se podía usar un JScrollPane y añadirle la tabla para que cuando esta sea muy grande, se pueda visualizar un scroll.

3. Implementación:

Logica:
En este paquete se encuentra la parte lógica(controlador) del programa que implementa el algoritmo goloso para resolver el problema del campeonato de fútbol.

Clase Torneo:

Se encarga de implementar el algoritmo goloso. Es la clase principal e implementa serializable. Sus variables de instancia son:

- ArrayList _fechas: un ArrayList de fechas para poder tenerlas enumeradas.
- ArrayList _equipos
- int _cantEquipos: inicializa la cantidad de equipos que tendrá el torneo.
- Map > _arbitraje: toma como key un Integer que representa a un árbitro y como value un map con equipo como key y un Integer cómo value que representa la cantidad de veces que el árbitro dirigió al equipo.
- Map> _historialDeArbitro: relaciona un árbitro como Integer con una lista de partidos.

Torneo(ArrayList fechas, ArrayList equipos):
Constructor de la clase Torneo. Toma como parámetros un ArrayList de fechas y otro de Equipos. En este se inicializa las variables de instancia fechas y equipos, y se asegura que las fechas pasadas como parámetro sean válidas, es decir, que ningún elemento del ArrayList sea null.

agregarArbitro():

Agrega un árbitro al Map arbitraje asignándole un número que lo identifique, siendo este número el tamaño de arbitraje más uno al momento de agregar el árbitro. También se agrega un HashMap a arbitraje

cambiarNombreAArbitro(int idArbitro, String nombre):
Toma como parámetro un árbitro y el nombre que se desea poner. Se asegura que el árbitro exista, luego si existe recorre el historialDeArbitro y por cada partido en el que aparezca el árbitro pasado como parámetro se setea un nuevo árbitro con el nombre pasado por parámetro.

mezclarArbitraje():
Hace que el arbitraje sea aleatorio para cada nuevo arbitraje. Mezclando a los árbitros con intercambiarArbitro(idArbitroA, idArbitroB).

intercambiarArbitro(idArbitroA, idArbitroB):
Primero valida que los árbitros dados por parámetro existan. Luego toma esos dos árbitros e intercambia sus historialesDeArbitro y por último se cambia el nombre del árbitro en cada partido para que se corresponda con los nuevos arbitrajes.

asignarArbitraje():
Se inicializa el arbitraje y el historialDeArbitro como un HashMap y se agregan árbitros a arbitraje con agregarArbitro(). Esto se hace en este método y no en el constructor para que cada vez que se llame se reinicie el arbitraje. También se asigna la el arbitraje de cada fecha con el algoritmo goloso que implementa asignarArbitrajeEnLaFecha(int fecha).

asignarArbitrajeEnLaFecha(int fecha)
En este método se implementa el algoritmo goloso. En el cual se recorre un HashSet de árbitros en el que están todos los árbitros de arbitraje y se saca el promedio de veces que dirigió cada partido. Al ID del árbitro se lo guarda en un HashMap junto con su promedio. Y luego, se ordena el HashMap promediosArbitros con sort de menor a mayor, para asignarles a cada árbitro de manera óptima el partido que le toca dirigir. -

validarIDArbitro(int idArbitro):
Tira una IllegalArgumentException si el ID del árbitro dado como parámetro es menor o igual a cero, o es mayor que el tamaño del HashMap arbitraje.

Clase Arbitro:

Se encarga de crear y guardar la información de un árbitro. Implementa Serializable y crea un HashCode y Equals para poder compararlo cuando sea necesario. Esta es su variable de instancia.

- String nombre

Arbitro(String nombre):
Constructor de la clase Arbitro. Toma como parámetro una String para inicializar el nombre y lanza una IllegalArgumentException si el nombre es null.

Clase Equipo:

Se encarga de crear y guardar la información de un árbitro. Implementa Serializable y crea un HashCode y Equals para poder compararlo cuando sea necesario. Esta es su variable de instancia:

- String nombre

Equipo(String nombre):
Constructor de la clase Equipo. Toma como parámetro una String para inicializar el nombre y lanza una IllegalArgumentException si el nombre es null.

Clase Partido:

Crea un partido con los equipos y el árbitro que participan en él. Implementa Serializable y crea un HashCode y Equals para poder compararlo cuando sea necesario. Estas son sus variables de instancia:
- Equipo equipoLocal, equipoVisitante
- Arbitro arbitro

public Partido(Equipo equipoLocal, Equipo equipoVisitante):
Constructor de la clase. Verifica que equipos sean válidos lanzando un IllegalArgumentException si son null o si el equipoLocal y el equipoVisitante son iguales. Luego, inicializa el equipoLocal y el equipoVisitante, y el árbitro se inicializa a null ya que el partido comienza sin árbitro hasta que se asigna cada árbitro a un partido con el algoritmo goloso de asignarArbitrajeEnLaFecha (int fecha) de Torneo

Clase Fecha:

Esta clase se encarga de guardar una lista de partidos. Implementa Serializable. Esta es su variable de instancia:

- ArrayList partidos

Fecha(ArrayList partidos):
Constructor de la clase Fecha. Lanza una IllegalArgumentException si la partidos dado por parámetro es null sino, inicializa this.partidos como el ArrayList de partidos dado por parámetro y le agrega los partidos con agregarPartidoAFecha(Partido partido).

asignarArbitroAPartido(int partido, Arbitro arbitro):
Primero verifica con validarIndicePartido(int partido) que el número de índice que se pasa por parámetro sea correcto y luego se setea el árbitro que se da como parámetro en el partido requerido.

Archivos:

Este paquete se encarga de la manipulacion de los archivos JSON

CalendarioJSON:
Es la clase que se encarga de todo lo relacionado a la manipulación de archivos JSON para el calendario. Implementa Serializable. Esta es su variable de instancia:

- ArrayList calendario

CalendarioJSON(ArrayList calendario):
Constructor de la clase CalendarioJSON en donde inicializa el ArrayLIst de calendario.

generarJSON():
Crea un nuevo Gson y lo convierte a JSON.

generarJSONPretty():
Crea un nuevo Gson en la forma Pretty, donde sus elementos están acomodados de forma que faciliten la lectura, y lo convierte a JSON.

guardarJSON(String jsonParaGuardar, String archivoDestino):
Guarda el contenido del jsonParaGuardar en archivoDestino.

leerJSON(String archivo):
Lee el contenido del archivo JSON que se encuentra en la ruta dada como parámetro.

Clase EquiposJSON:

Es la clase que se encarga de todo lo relacionado a la manipulación de archivos JSON para el listado con los equipos del torneo. Implementa Serializable. Esta es su variable de instancia:

● ArrayList listaDeEquipos

EquiposJSON(ArrayList calendario):

Constructor de la clase EquiposJSON en donde inicializa el ArrayLIst con la lista de equipos.

generarJSON():

Crea un nuevo Gson y lo convierte a JSON.

generarJSONPretty():

Crea un nuevo Gson en la forma Pretty, donde sus elementos están acomodados de forma que faciliten la lectura, y lo convierte a JSON.

guardarJSON(String jsonParaGuardar, String archivoDestino):

Guarda el contenido del jsonParaGuardar en archivoDestino.

leerJSON(String archivo):

Lee el contenido del archivo JSON que se encuentra en la ruta dada como parámetro.

Vista:

En este packpage se encuentra todo lo relacionado a la parte gráfica. En donde se va a generar un panel por fecha para mostrar cada partido con su arbitraje. VentanaPrincipal: La ventana principal de la parte gráfica, en donde se va a controlar el funcionamiento de todos sus paneles. Estas son sus variables de instancia: ● JFrame frame: el frame principal de la ventana.

● HubPanel hub: el panel en donde se encuentran los controles de la interfaz gráfica del programa.

● ArrayList calendarioPaneles: ArrayList de FechaPanel en donde se guardan paneles con distinta fecha. 12

● int indiceDePagina: sirve para controlar en qué página del calendario se encuentra.

VentanaPrincipal():

En donde se inicializa un nuevo HubPanel y se llama a initialize() para inicializar el frame principal.

mostrar():

Hace visible el frame principal.

cargarTorneo(ArrayList calendario):

En esta clase se cargan los partidos del torneo al ArrayList de calendarioPaneles, se inicializa el índice de página en cero y se setean los botones para moverse entre páginas para que el btnFechaAnterior no se pueda clickear ya que al empezar por la primera página no habría una anterior, y el btnFechaSiguiente si se pueda clickear.

paginaAnterior():
Primero pregunta si existe una página anterior con existePaginaAnterior(). Sí existe, se llama a cambiarDePagina(int indiceDePagina), se resta uno al indiceDePagina y se habilita el btnFechaAnterior, en caso contrario se deshabilita el btnFechaAnterior.

paginaSiguiente():
Primero pregunta si existe una página siguiente con existePaginaSiguiente(). Sí existe, se llama a cambiarDePagina(int indiceDePagina), se suma uno al indiceDePagina y se habilita el btnFechaSiguiente, en caso contrario se deshabilita el btnFechaSiguiente.

existePaginaAnterior():
Retorna true si el indiceDePagina es mayor a cero sino retorna false.

existePaginaSiguiente():
Retorna true si el indiceDePagina es menor al calendarioPaneles.size()-1.

getBtnAsignarArbitraje():
Obtiene el boton btnAsignarArbitraje de la clase HubPanel.

getBtnCargarArbitro():
Obtiene el botón btnCargarArbitro de la clase HubPanel.

getBtnFechaAnterior():
Obtiene el botón btnFechaAnterior de la clase HubPanel.

getBtnFechaSiguiente():
Obtiene el botón btnFechaSiguiente de la clase HubPanel.

Clase HubPanel:
Esta clase se encarga de crear todos los botones que se usarán en el programa. Estas son las variables de instancia:
- JButton btnFechaAnterior
- JButton btnAsignarArbitraje
- JButton btnFechaSiguiente
- JButton btnCargarArbitro

HubPanel():
Constructor de la clase. En él se inicializan los JButton btnFechaAnterior, btrnAsingnarArbitraje, btnFechaSiguiente, btnCargarArbitro y se crea un JLabel.

FechaPanel:
En esta clase se crea la parte de la interfaz gráfica donde se mostrará el calendario con las fechas y los partidos del torneo. Estas son sus variables de instancia: ● JTable tabla: la tabla donde se muestran los partidos.

- JLabel cartelFecha
- Color COLOR_CARTEL_FECHA= new Color (35,190,184): inicia un color para luego dárselo al cartelFecha.
- Color COLOR_BORDE_CARTEL_FECHA=new Color (2,163,156)
inicia un color para luego dárselo al cartelFecha.
- Color CABECERA_COLOR= new Color(109,117,205):

inicia un color para dárselo a la primera fila de la tabla. ● JScrollPane scroll

FechaPanel(String fecha):
Constructor de la clase FechaPanel. Crea una nueva JTable, le da una posición en la pantalla y le agrega color y un tamaño adecuado con propiedadesTabla(). También crea un nuevo JScorllPane asociandole la tabla para que pueda recorrerla y mostrar cada parte de la tabla. Por último crea un Jlabel y le agrega propiedades con propiedadesCartel().

cargarFechaEnTabla(ArrayList partidos):
Carga a la tabla el calendario con los partidos, los árbitros, si es que ya están decididos por el programa, y los equipos del torneo. Luego agrega a la tabla propiedades gráficas con propoiedadesTabla();

mostrar(boolean visibilidad):
Toma un boolean como parámetro para decidir si hacer visible todo el panel de FechaPanel o no.

VentanaCargarArbitro:
Esta clase es hija de JInternalFrame para crear una nueva ventana que no pueda salir del panel en uso y sirve para darle un nombre al árbitro deseado. Sus variables de Instancia son:
● JTextField nombreArbitro
● JSpinner idArbitro ●
JButton btnCargar, btnCancelar

VentanaCargarArbitro():
Constructor de la clase VentanaCargarArbitro. En él si inicializan todos elementos de éste nuevo frame y se le dan propiedades apropiadas.

Clase UIMain:
Clase que se encarga de tomar las acciones necesarias en relación al input del usuario para el funcionamiento de la parte gráfica del programa.
Estas son sus variables de instancia:

● VentanaPrincipal ventana
● JButton btnFechaAnterior
● JButton btnAsignarArbitraje
● JButton btnFechaSiguiente
● JButton btnCargarArbitro
● JButton btnCargarEnVCA
● JButton btnCancelarEnVCA
● Torneo torneo
● boolean primerClicEnBtnArbitro: para controlar que el boton de asignar árbitro se haya apretado y así poder mezclar el arbitraje si es que ya se ha apretado el botón.

UIMain(VentanaPrincipal ventana):
Constructor de la clase UIMain. Inicializa todos los botones que tiene el programa con el método inicializarBotones(), la ventana principal e inicia el Torneo en null para hacerlo en otro método. También mediante el método agregarEventos() se agregan los eventos de todos los botones.

mostrarVentanaPrincipal():
Con el método mostrar() de la clase VentanaPrincipal hace visible el frame principal del programa.
cargarCalendarioAlTorneo(ArrayList calendario, ArrayList equipos):
Es un método que carga el calendario del torneo al programa. Esto se hace aparte y no en el constructo porque en este caso se decidió que se lea desde un archivo JSON, pero si el calendario llega a estar en otro tipo de archivo se rompería el código.

Conclusión:

El algoritmo goloso siempre devuelve la misma solución pero si este mismo algoritmo hubiese tomado la decisión de seleccionar un árbitro aleatorio (entre los que tengan el mismo promedio) hubiese llegado a lo mismo que implementamos nosotros, el algoritmo tendría mayor complejidad y además sería más difícil testear .