Trabajo Práctico grupal
Programación III

Segunda Entrega

Integrantes:

Firmani, Gregorio
Etchepare, Mateo
Deserti, Manuel
Bedini, Tomas

Fecha de entrega: 26/06

Trabajo Práctico grupal Programación III
Segunda entrega

Puesto que es la segunda entrega, procederemos a explicar las novedades de la nueva versión de nuestra aplicación de búsqueda de trabajos, dando por entendido que se tiene en cuenta lo realizado en la primera parte del trabajo práctico.

- **Simulación con Threads:** utilizando concurrencia llegamos a realizar una simulación con threads, siendo estos los empleados. El proceso se realiza en la clase "BolsaTrabajo", allí aplicamos 2 métodos synchronized, uno llamado "sacarTicket" que retorna el ticket acorde a las necesidades del empleador, retornando null en caso de que no se encuentre, mientras que el otro método es un void, que se lo invoca en el momento que la locación del empleador y la del empleado no coinciden y su función es introducir nuevamente el ticket simplificado en el arraylist de tickets simplificados. La razón por la que decidimos que sean synchronized los métodos es porque el recurso compartido no puede ser utilizado por 2 hilos de ejecución en simultáneo.
  Los métodos faltantes para lograr la simulación son el "run" implementado en la clase "Empleado" (la cual implementa Runnable), el método se ejecuta hasta que se den 10 repeticiones del mismo o el empleado encuentre un empleo que satisfaga sus requerimientos.

  Por otro lado, empleador implementa "eleccionTicketSimp", este método es el encargado de realizar todos los llamados y a su vez es el método al que recurre "run" para comenzar cada ciclo del thread.

  Los tickets simplificados son creados por el empleador (ya que es el que está buscando empleados), el cual puede hacer hasta 3 máximo y son llevados al arraylist de la bolsa de trabajo, a partir del patrón **observer-observable** el empleador mantiene una referencia al mismo, y en caso de que el empleado coincida con alguno de los tickets de búsqueda que éste propone, a partir del "notifyObservers" que se lanza, el empleador se entera que se contrató un empleado y en caso de tener el máximo de tickets posibles creados, puede realizar uno más.

  En la ventana del empleado se podrá elegir el rubro y la locación que éste desee en la búsqueda de la bolsa, pero por defecto se le inicializan valores preestablecidos para que cuando se llame a tal ronda de elecciones puedan participar de la misma.

Cabe aclarar que la bolsa de trabajo, así como los tickets y variables propias a este procedimiento no se serializan, por lo que cada vez que se inicie el programa se deberán crear nuevas instancias de ticket simplificado y fijar nuevamente las condiciones que tienen los empleados para con la búsqueda en bolsa de trabajo.

- **Utilización del patrón State:** a partir de una interfaz "I_EstadoTicket" se explicitan los métodos que implementan los objetos de los estados. Dentro del paquete "estadoTicket" se encuentran estas clases, estas son: activa, suspende, cancela, finaliza. Funcionan de modo tal que retorna un string para avisar el cambio de estado o la imposibilidad del mismo, ya que no es trivial el cambio de estos sino que sigue una correspondencia, la cual resulta ser:

  Activo: indica que el ticket se encuentra vigente, de este estado puede pasar a suspendido, cancelado o finalizado. En este estado puede cumplir las funcionalidades propias de la agencia, como la ronda de contrataciones y de elecciones.

  Suspendido: este estado indica la suspensión del ticket para no formar parte de los llamados de la agencia pero puede ser reactivado nuevamente cuando el usuario lo desee. Puede pasar a los estados activado y cancelado, pero no a finalizado.

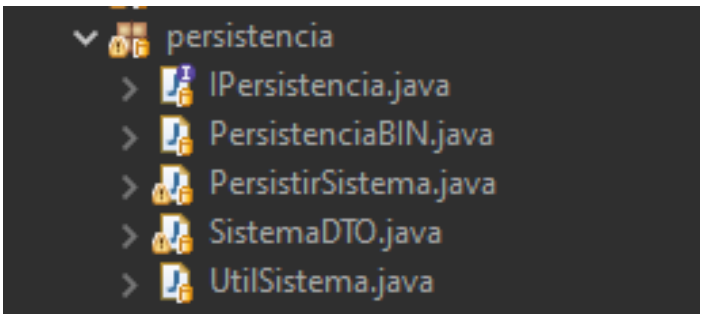
  Finalizado: Se presenta cuando el usuario logró su cometido, es decir si es empleado logró ser contratado y si es empleador logró obtener la n cantidad de empleados que sugirió en su ticket, de este estado no se puede volver, es decir no puede volver a cambiar.

  Cancelado: cuando el usuario no desea más poseer el ticket, recurre a cancelarlo, en este estado, al igual que en finalizado, no puede cambiarse a otro.

- **Persistencia**: La persistencia se logra a partir de varias clases incluidas en el paquete persistencia. Optamos por implementar serializable en todas las clases que buscábamos serializar menos en la clase "Sistema", la cual implementa un singleton, por lo que tuvimos que aplicar un patrón **DTO** para hacer la serialización de la misma.

  SistemaDTO es una de las clases de nuestro nuevo paquete, que también incluye una interfaz "IPersistencia" que tiene los métodos necesarios para realizar el guardado de datos, "PersistenciaBIN" implementa IPersistencia y contiene los métodos para hacer la lectura y escritura de los datos en el archivo binario.

"UtilSistema" y "PersistirSistema" contienen métodos estáticos que se utilizaran en el proceso. La primera permite hacer la transferencia de datos del sistema a la clase DTO y viceversa. Mientras que la segunda será la que de las pautas para realizar la persistencia.



La persistencia se da a partir de 2 botones indicados en el apartado de agencia de la ventana:

Guardar agencia: se utiliza para hacer la escritura en un archivo binario de los datos utilizados en el curso de la aplicación.

Recupera agencia: se utiliza cuando se ingresa a la ventana, sirve en caso que se quiera hacer la carga de datos de una anterior escritura en un archivo binario.

- **Parte de la interfaz gráfica (patrón MVC):** Para esta sección, detallaremos las partes más significativas, dado que en sí, en la clase de la ventana, hay muchas líneas de código. Dividimos las clases en dos paquetes: el paquete de **controlador** y el paquete de **vista**. La clase de vista va a tener una referencia al controlador, y el controlador va a tener una referencia al modelo y otra referencia a la ventana. Toda la lógica de prender-apagar botones, evitar realizar acciones si el usuario no está logueado, y similares, está programado en la clase de ventana, ya que ahí implementamos métodos para escuchar los campos de texto y para realizar ciertas funciones específicas en los botones (por ejemplo, vaciar campos de texto). Para esto último, implementamos un método de MouseListener en la propia ventana, pero sólo para intervenir en lo mencionado anteriormente, no para modificar parte de datos.

  El controlador, por su parte, implementa ActionListener, y allí desembocan todos los eventos que genera presionar los botones de la ventana. El controlador busca los datos, y los vuelca en la ventana.

La ventana se diseñó con un TabbedPanel, así incluímos la parte de Agencia, Empleado, Empleador y la parte de simulación de threads, todo en una misma ventana. Sin embargo, utilizamos ventanas emergentes para informar ciertos errores del sistema. El sistema de login hecho desde la ventana consiste en crear un usuario y luego que el usuario se tenga que loguear, así como hacen la mayoría de los sitios web.

Breve explicación de cómo funciona el programa:

Cuando ingresamos lo primero que se verá será la vista de la Agencia/Sistema, en ella podremos ponerle un nombre a la agencia solamente una vez, ya que implementa el patrón singleton.

En caso que ya tengamos una empresa y queramos obtener los datos de la misma, debemos presionar "Recuperar agencia", que lee el archivo binario guardado anteriormente. También se puede guardar lo trabajado presionando en "Guardar agencia".

Continuando con esta ventana, en el apartado de las funcionalidades de ésta, nos encontramos (además de con la parte de serialización explicada anteriormente) con 3 botones:

- Ronda de elecciones: ya realizado en la entrega anterior, permite ponderar los usuarios respecto de otros.
- Ronda de contrataciones: se permite presionar luego de ser llevada a cabo la ronda de elecciones, es el último paso para que se den las contrataciones.
- Bolsa de trabajo: cuando se lo presiona, cambia la pantalla a la tabpannel de "Bolsa de trabajo" y comienza la simulación con threads entre los tickets simplificados y los empleados.

Los otros apartados de las ventanas permiten mostrar los campos que dicen arriba a la izquierda, como "Contrataciones", "Lista de Empleados", etc.

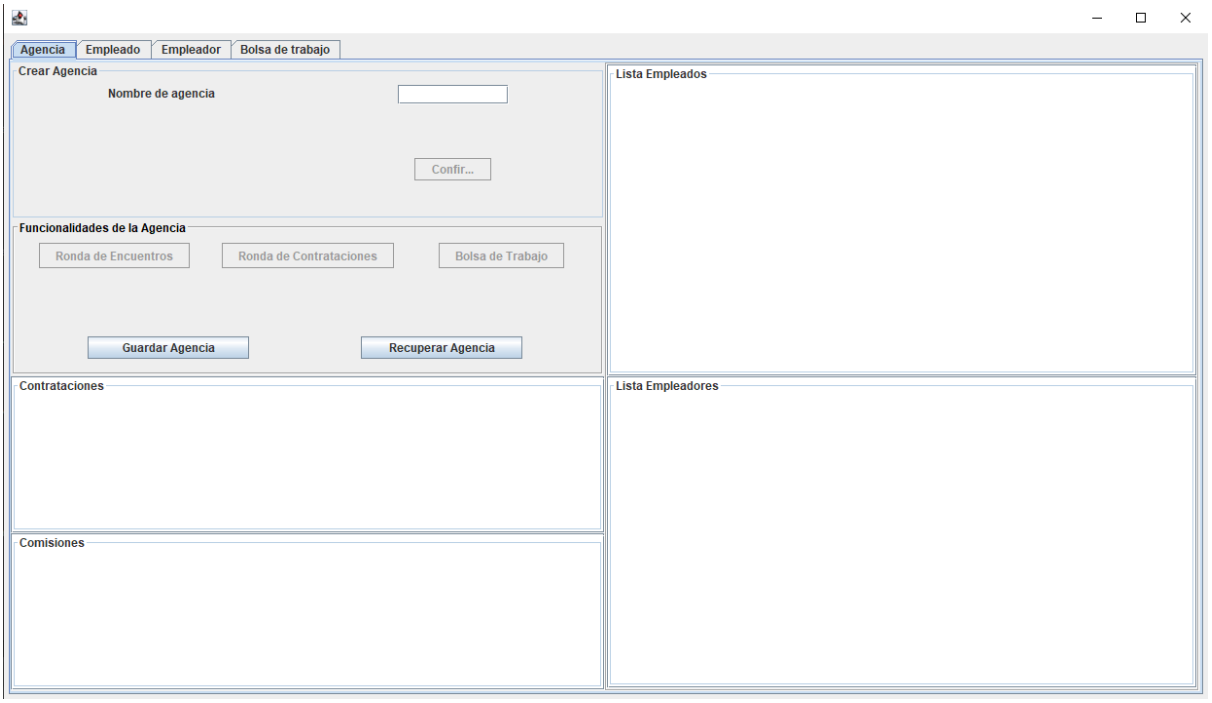
La vista del Empleado permite a los empleados crear su usuario, luego loguearse, crear un ticket, su estado y verlo; por otro lado, tiene un apartado para aplicar en la ronda de elecciones, y debajo de eso, permite elegir su situación para la bolsa de trabajo (como estaba especificado, se establecen valores por defecto para hacer que interactúe en la bolsa de trabajo).

La vista del Empleador es bastante similar a la del empleado, una zona para loggin, una para crear su usuario, una vez logueado puede crear su ticket de búsqueda y también puede seleccionar al empleado en la ronda de elecciones. Debajo de ronda de elecciones tiene un apartado para crear su ticket simplificado, puede crear tener hasta 3 vigentes.

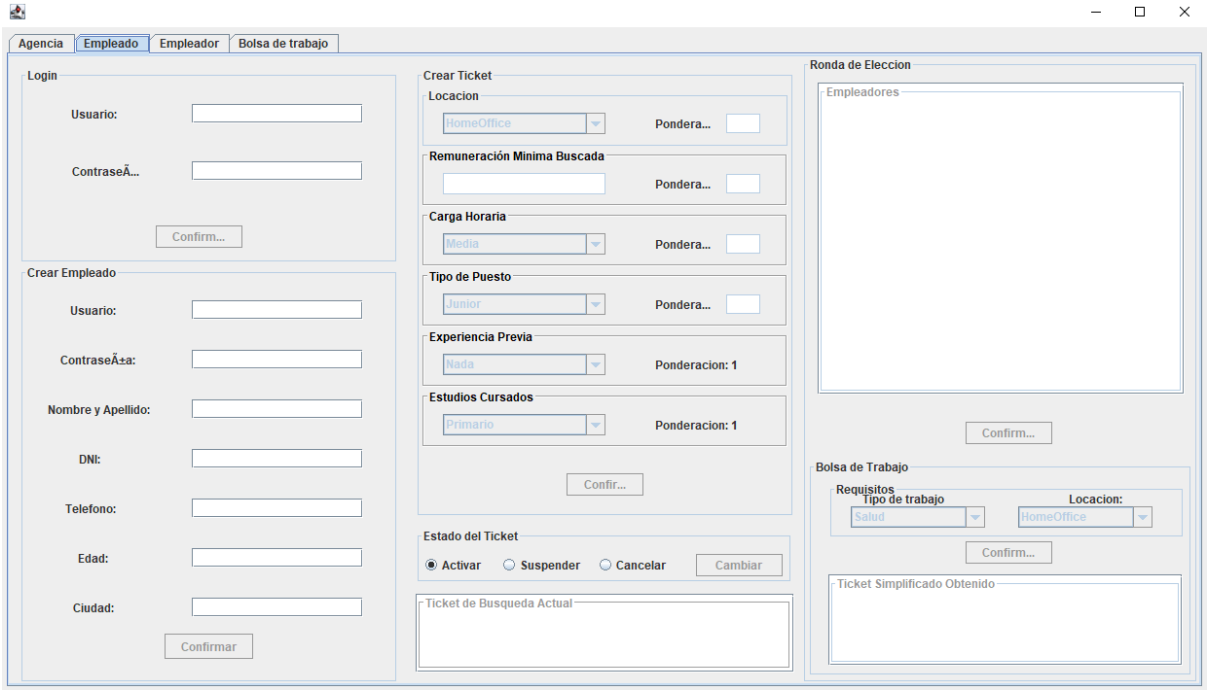
La vista de la Bolsa de Trabajo que realiza una simulación por threads, permite ver los tickets simplificados que se encuentran en la bolsa de trabajo en la primer columna, la simulación como tal en la segunda, y las contrataciones en la tercera.

Imágenes de las ventanas:

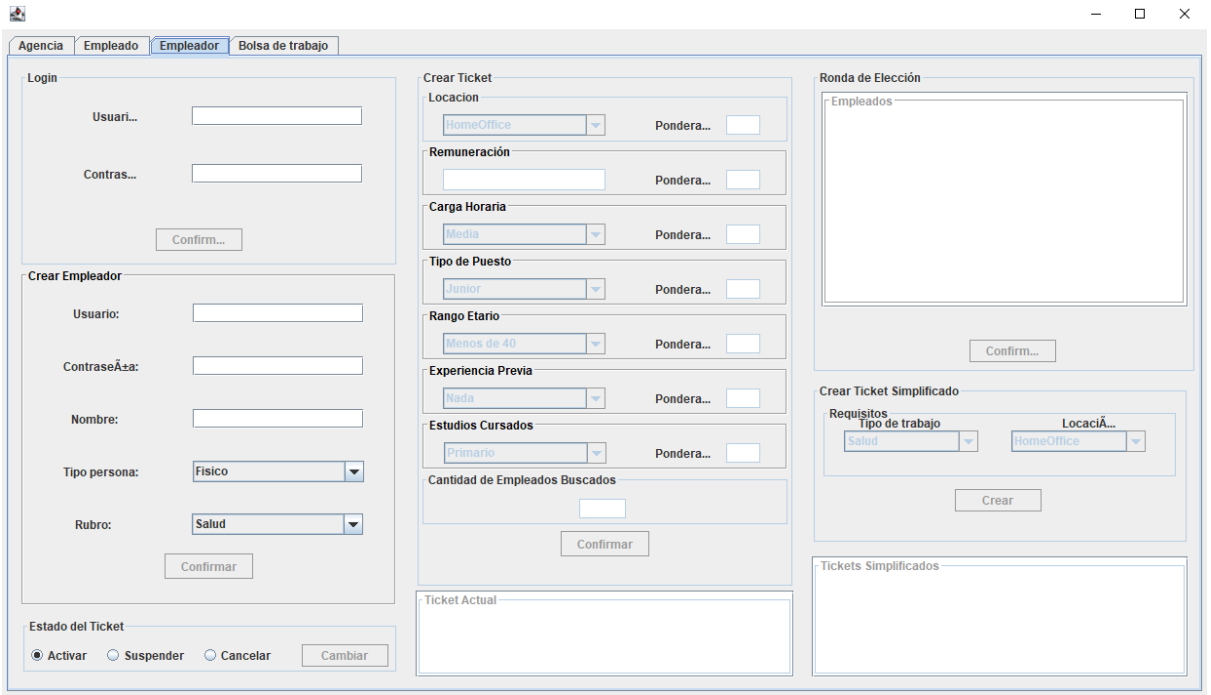
- Vista de la Agencia/Sistema:



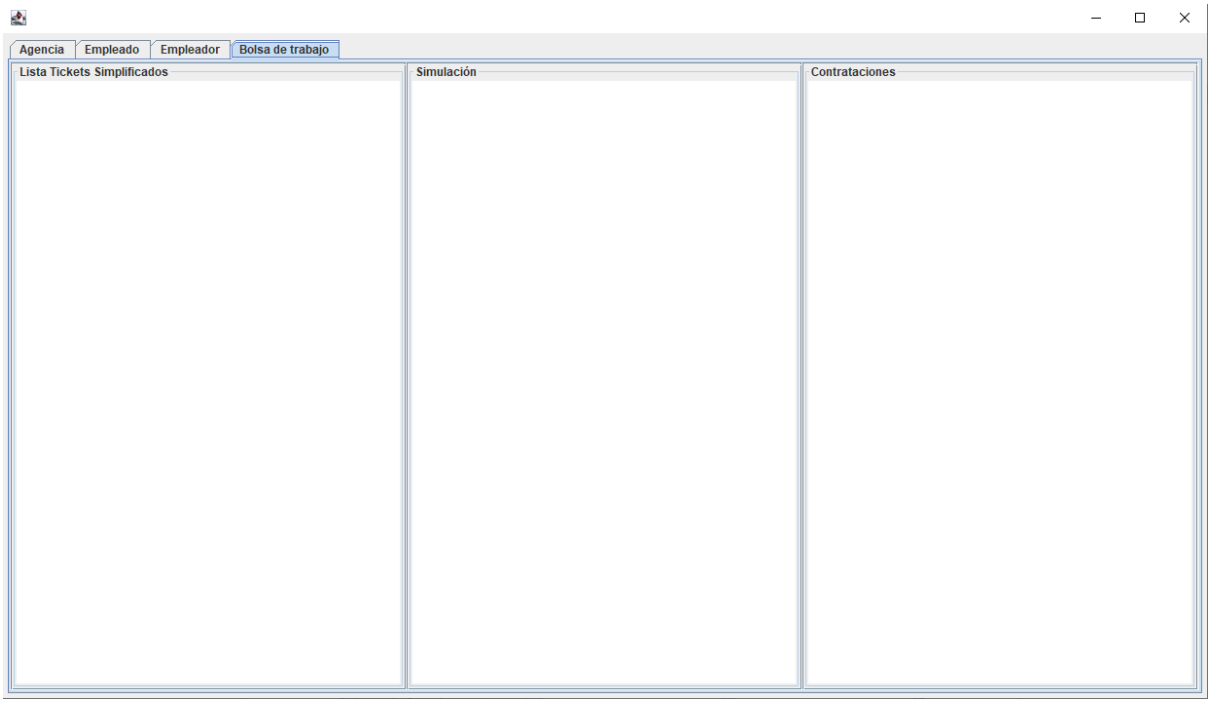
- Vista del empleado:



- Vista del empleador:



- Vista de la sección de simulación por threads (Bolsa de Trabajo):



Conclusión

En conclusión, y para cerrar el trabajo, al igual que en la primera entrega, este trabajo nos permitió afrontar un desafío que puede ser que tengamos que transcurrir el día que estemos transitando nuestra vida profesional. Supimos implementar y aplicar los temas aprendidos en esta segunda mitad de cuatrimestre.

Globalmente destacamos que pudimos desarrollar la capacidad de poder trabajar en grupo, aplicando una comunicación fluida, compartiendo ideas y escuchando al compañero para poder superar los problemas que se nos dieron.

Repositorio GitHub:

https://github.com/Greg1704/Progra3TpCursada.git