Informe TP Grupal
Programación III

Fecha: 26/06/2022

GRUPO 9

PARTICIPANTES:
Garcia, Ezequiel
Gonzalez, Franco
Vicente Juan Martín

PROFESORES:
Gellon, Ivonne
Guccione, Leonel
Lazzurri, Guillermo

1. Objetivo

El objetivo del trabajo práctico grupal es la aplicación de los conceptos teórico-prácticos adquiridos en la cátedra, para la realización de un sistema para la gestión de búsquedas laborales. Aplicar estos nuevos conocimientos relacionados a la programación orientada a objetos (clases y sus relaciones, herencia, polimorfismo), patrones de diseño, excepciones, serialización, concurrencia, interfaces gráficas y desarrollo de proyectos en el lenguaje de programación “Java”.

2. Alcance

El sistema realizado tiene como alcance las “Consideraciones Generales sobre el Trabajo Práctico Grupal” otorgado por la cátedra y las consideraciones charladas en las clases correspondientes.

3. Desarrollo de Actividades

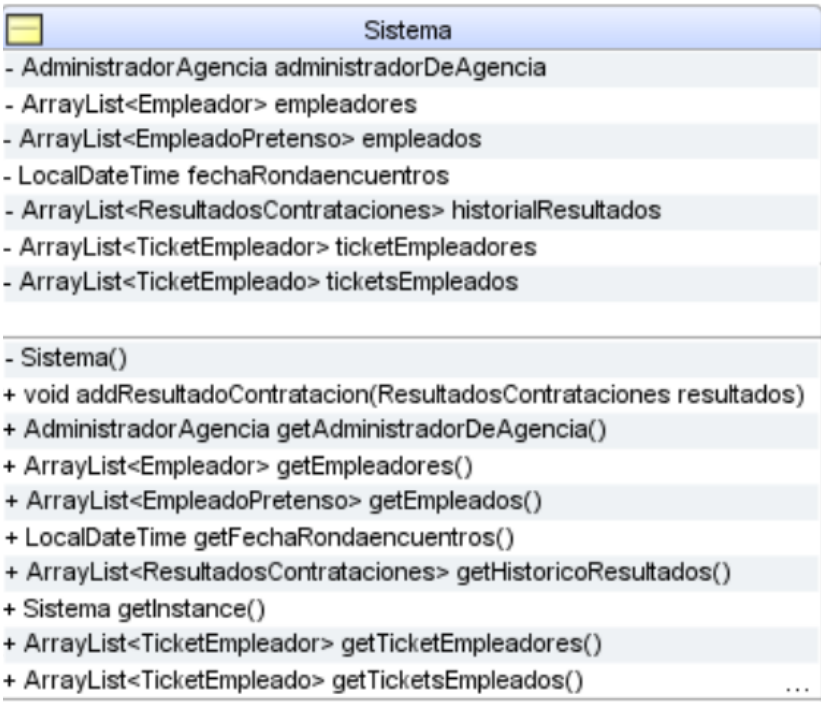
Las tareas desarrolladas en el presente trabajo se encuentran en los siguientes anexos:

- Proyecto Java:
  https://github.com/Franco-15/Trabajo-Practico-Progra3

4. Descripción simplificada del trabajo realizado

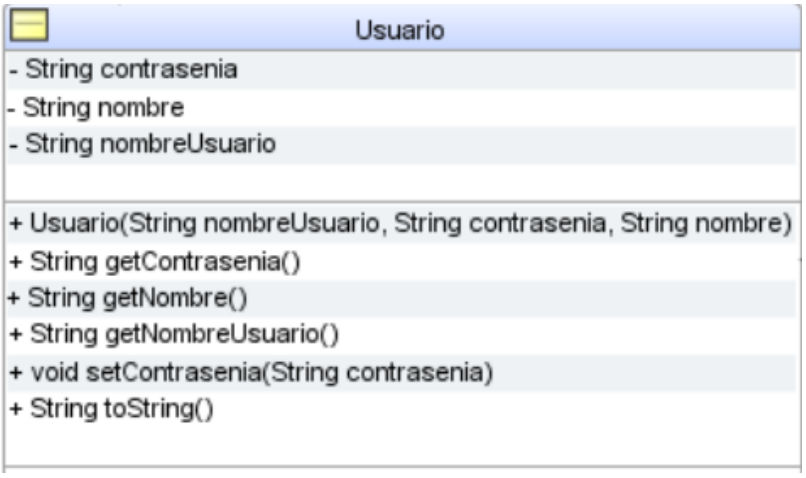
● Sistema

Se generó una clase sistema contenedora (utilizando el patrón Singleton) que contiene a los objetos y características del programa que permitirán la correcta ejecución de cada una de las tareas.



- Usuarios

Allí se permite la generación de nuevos usuarios, sean Empleados Pretensos, Empleadores Físicos, Empleadores, almacenando los mismos en el ArrayList correspondiente, comprobando que no exista otro usuario del mismo tipo con el mismo nombre de usuario. También permite la generación de un Administrador del Sistema.
Para la generación de usuarios se ha implementado el Patrón Factory.



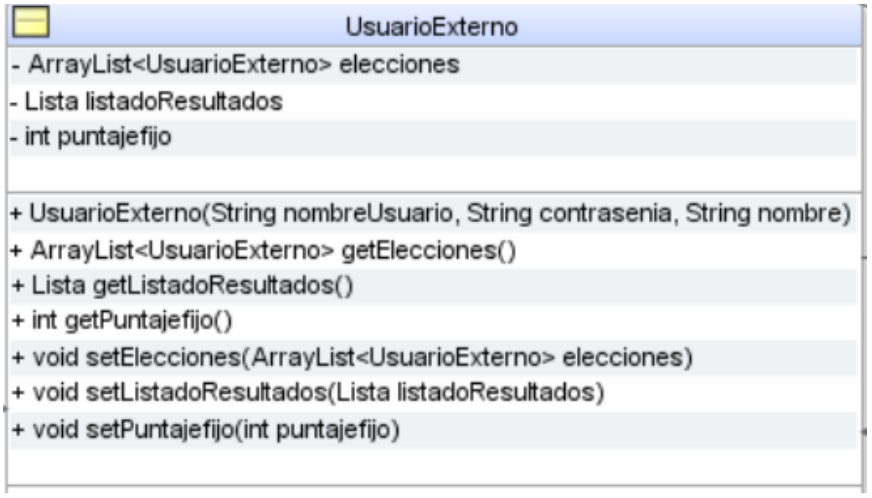
- Generación de Tickets

Los usuarios pueden generar el Tickets para poder participar en la próxima “Ronda de Encuentros Laborales”. Si bien los Tickets son diferentes para empleados pretensos y empleadores potenciales, los mismos son generados por un único método gracias al patrón Factory.

Estos Tickets dan a los usuarios la posibilidad de cargar sus solicitudes con respecto a las características del empleo ofrecido y la cantidad, para el caso de los empleadores, y el empleo buscado para los empleados pretensos.
También el usuario puede gestionar el estado de los tickets cargados, asignando el estado de activo, suspendido o cancelado.

- Ronda de Encuentros Laborales

Función propia del administrador, aquí se elaborarán las listas de resultados de cada usuario externo (como se verá en la figura de más abajo), con la condición de que tenga un ticket generado y el mismo se encuentre activo. Además se actualizará la fecha de esta misma ronda que se encuentra como atributo en la clase sistema. Mediante un recorrido de todos los usuarios externos, se actualizan sus respectivas listas con las respectivas fechas de cada una y ordenadas por su puntaje con dicho usuario. dichas listas se instancian cuando se crean los usuarios , nunca serán nulas sino que siempre podrán contener información o simplemente estar vacías y colocando la respectiva fecha al momento de crear el usuario o al momento de actualizar dicha lista en esta misma ronda.



- Ronda de Elecciones

El programa permite a los usuarios visualizar los resultados obtenidos en la ronda de contrataciones, ordenados de mayor a menor por puntaje. Con esta información, los diferentes usuarios pueden elegir la cantidad de empresas o empleados que desee, almacenando los mismos en un vector de tipo (UsuarioExterno) que será utilizado en la ronda de Contrataciones para comparar con las elecciones de los demás usuarios.

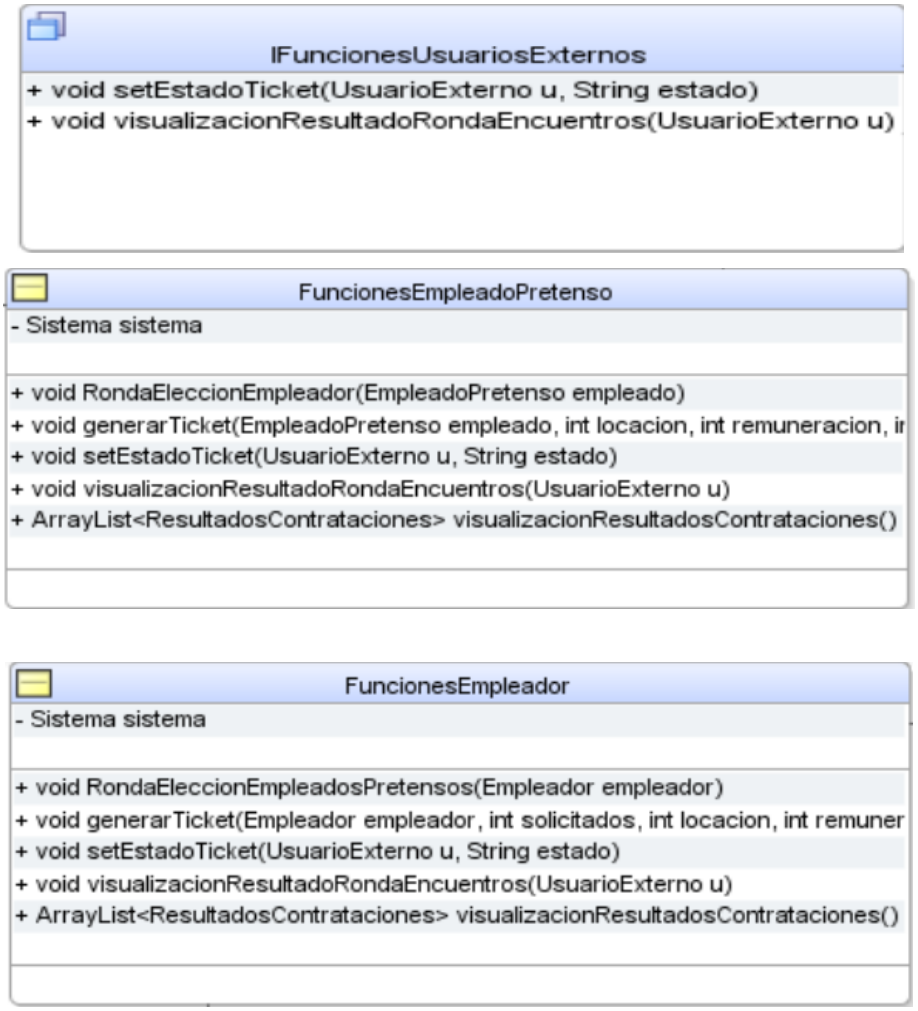
- Ronda de Contrataciones

Se desarrolla una función en este caso (en la clase funciones administrador) que recorre todo el arreglo de empleadores registrados en el sistema, y para cada uno lo primero que hace es comparar la fecha de la lista del respectivo empleador y la fecha de la última ronda de encuentros. si dicha fecha es menor a la última ronda, recorre todo el arreglo de usuarios

elegidos (atributo de empleadores), si dicho usuario tiene en su arreglo de elegidos (atributo en empleados) al mismo empleado, existirá un “match”. si ese fuera el caso, el administrador cobra sus respectivas comisiones, se limpia el arreglo de elecciones del empleado y se establece su ticket en “finalizado”, y si el empleador consiguio el numero de empleados solicitados (atributo empleados solicitados , empleados obtenidos en la clase empleador), se establece también como finalizado el ticket de dicho empleador. También existirá un registro de estas contrataciones en un Array List perteneciente al sistema que permitirá almacenar las contracciones históricas.

- Visualización Resultado Ronda Encuentros

La función que desarrollan los usuarios, se encuentra en la interfaz “iFuncionesUsuarios Externos”, que la implementan tanto la clase “funciones empleador” y la clase “funciones empleado pretenso”. Cada clase desarrollará el método ya que el parámetro de entrada de la función desarrollada es un usuario externo y cada clase deberá castear según la clase que lo esté invocando y mostrará los resultados acordes.



- ## Visualización Resultado Contrataciones

Al final de la ronda de contrataciones, se visualizan los resultados obtenidos en caso de existir contrataciones Por otro lado, también existe un registro histórico de todos las contrataciones efectuadas en todas las rondas de contrataciones hasta el momento (atributo arreglo en sistema). Este arreglo será de tipo ResultadosContrataciones, que contiene los datos de quien contrato a quien y la fecha de la misma contratación.
Como cada ronda tiene su propia fecha, se podrá visualizar las contrataciones efectuadas con sus respectivas fechas.

- ## Aplicación de Interfaces Gráficas

Se crearon las ventanas del sistema que permiten el trabajo sobre el mismo, teniendo una ventana principal que permite ingresar a registrar un nuevo usuario según su clasificación y luego también a cualquier tipo de usuario ingresar a las funciones que le corresponden dependiendo de las tareas que ese tipo de Usuario pueda realizar.

- ## Serialización

Se diseñó un sistema que permite la persistencia de los datos a través del tiempo. El mismo consiste en la generación de un archivo binario en el que se copian todos los datos registrados en el sistema, y la lectura de ese archivo al ejecutar el programa y arrancar a trabajar con Usuarios, tickets, Rondas e historiales de contrataciones que ya se encontraban dentro del sistema.

- ## Concurrencia

Se realizó una simulación con tickets simplificados que emulan una bolsa de trabajo donde los empleadores dejan ticket con el rubro en el cual se desempeñan y el tipo de locación pretendido, y los empleados pretensos van a buscar un ticket conforme a sus expectativas, los mismos pueden buscar solo 10 veces en la bolsa.
Como entendemos que esta parte del trabajo se realiza para demostrar los conocimientos aprendidos pero no siguen los lineamientos generales del trabajo, se realizaron clases aparte. También, para su resolución se buscó la eficiencia dentro de los métodos sincronizados para que cada hilo se encuentre lo menos posible utilizando los mismos.

5. Patrones de Diseño Utilizados

Como hemos estudiado a lo largo del cuatrimestre, la utilización de patrones de diseño tienen diferentes ventajas que nos pueden ayudar a la resolución de nuestro trabajo práctico. Los patrones utilizados para el desarrollo del trabajo fueron los siguientes:

- Singleton
- Factory
- Double Dispach
- MVC
- Observer Observable
- DAO DTO

PATRON SINGLETON

Nuestra clase sistema es la que va a contener todas las otras clases desarrolladas en nuestro sistema, esto implica que debe instanciarse una única vez. Por norma general, esta clase es accesible de forma global, y el proceso de instanciado no requiere parámetros.

Esta clase Sistema por contar con este patrón presenta:

- Un método Instance() de carácter estático (método de clase) encargado de instanciar la clase.
- Un constructor privado que evita que se creen nuevos objetos mediante new(), haciendo que el método Instance() sea el único que puede generar la instancia.
- La clase Sistema cuenta con un atributo de la propia clase de carácter privado.
- El constructor (privado) se encarga de construir el objeto.
- El método estático Instance() realiza dos operaciones: Comprobar si el atributo es null. En ese caso, se invocará al constructor. En caso contrario, se devolverá el objeto existente.

El objetivo de la utilización de este Patrón es contar con una sóla instancia de clase "Sistema", y esta instancia será la contenedora de todas las instancias y relaciones que se generen dentro de nuestro programa desarrollado.

PATRON FACTORY

Provee uno de los mejores modos de crear un objeto. En el patrón Factory, se crea un objeto sin exponer la lógica de la creación al cliente y retorna una referencia al nuevo objeto creado, usando una interfaz común.
Los patrones creacionales o de creación son aquellos en los que se delega la instancia de un objeto en otro, en lugar de recurrir a un simple new(). Esta forma de trabajar puede ser útil en algunos escenarios, pero el principal suele involucrar el no saber qué objeto vamos a instanciar hasta el momento de la ejecución. Se utiliza el polimorfismo para utilizar una interfaz para alojar una referencia a un objeto que será instanciado por un tercero en lugar de dejar que sea el propio constructor del objeto el que proporcione la instancia. Por tanto, nuestro objetivo principal será la encapsulación de la creación de objetos.
Este patrón es utilizado en la creación de Usuarios, con una sola función estática se pueden generar usuarios de tres tipos, Empleados Pretensos, Empleadores Jurídicos y Empleadores Físicos. Lo mismo realiamos con los Tickets, que con un sólo método se pueden instanciar Tickets de Empleados Pretensos y Empleadores.

PATRON DOUBLE DISPACH

Éste método se utiliza para modelar una interacción cuando el resultado depende del receptor y del argumento. En vez de escribir código que específicamente verifica la clase

del parámetro, agregar nuevos métodos que tengan el mismo nombre (un método secundario) para cada clase de todos los potenciales objetos parámetros.
Escribir el método original para simplemente llamar este nuevo método secundario, pasando el receptor original como argumento. Es responsabilidad de cada objeto que recibe este mensaje secundario conocer qué debe hacer. Típicamente, cada método secundario invoca una operación específica sobre el receptor original.
Este patrón se aplica en la generación de la puntuación en el enfrentamiento de tickets al momento de realizar la “Ronda de Encuentros Laborales”. Nosotros lo utilizamos para realizar la comparación de las elecciones de locación. Sea cual sea la que elija el empleador pretenso (Home Office, Presencial o Indistinto), se va a generar una instancia del tipo correspondiente y con ella va a llamar al método que le asigne el valor a la puntuación de ese enfrentamiento.
Este patrón se podría aplicar en todas las comparaciones del trabajo, pero eso implica que realizar 24 clases adicionales. por esta razón, decidimos trabajar mediante arreglos estáticos que asignen los valores correspondientes en cada enfrentamiento.

MVC MODELO VISTA CONTROLADOR
El MVC es un patrón de arquitectura de software, que separa los datos y principalmente lo que es la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.
Nosotros en la primer parte del trabajo desarrollamos el modelo del negocio, y en esta segunda parte se diseñaron los controladores y las vistas correspondientes para la correcta ejecución del programa.

OBSERVER OBSERVABLE
Para las interfaces gráficas se utilizó el patrón Observer Observable para poder mostrar en los cambios que realizan los usuarios en algunas de sus funcionalidades.
El observable (heredando de una clase abstracta) mantiene una colección de los observadores. Cada vez que sucede algo significativo para este objeto emite una notificación a todos los observadores. Esto provocará que se ejecute un método en cada observador
El observador mantiene una referencia de cada objeto observado (puede ser más de uno). Implementa el método update (propio de la interfaz que implementa) que se ejecuta cuando los observados notifican. Es responsabilidad del observador mantener vincularse con el observado.

DAO - DTO
Se utilizó este patrón para lograr la persistencia de los datos que se van incorporando al sistema.
Por cada fuente de datos se debe implementar un DAO específico para ella, sin modificar el resto del código: cualquier cambio en la fuente de datos implicará la utilización de un DAO distinto, pero la lógica de negocio permanecerá indiferente ante tal cambio. El DAO es una suerte de traductor entre el idioma que habla nuestra aplicación y el idioma que habla una fuente de datos específica (habrá un DAO para SQL Server, otro para MySQL, otro para archivos, etc.)

Esta implementación DAO se realizó a través de la clase UtilSistema, el cual permite copiar los datos necesarios de la clase Sistema en una clase Sistema DTO para persistir la información, y viseversa, para poder seguir utilizando esos datos.
Un DTO se conforma de una serie de atributos provenientes de una o más fuente de datos, la información puede ser pasada tal cual está o estar compuesta por más de un campo, también permite omitir información no requerida.
Características de un DTO
Si bien un DTO es simplemente un objeto plano, tiene que cumplir algunas reglas.
Solo lectura: ya que el objetivo de un DTO es ser utilizado como un objeto de transferencia entre el cliente y el servidor, es por ello que solo deberemos de tener los métodos GET y SET de los respectivos atributos del DTO.
Serializable: Es claro que, si los objetos tendrán que viajar por la red, deberán de poder ser serializables, pero no hablamos solamente de la clase en sí, sino que también todos los tributos que contenga el DTO deberán ser fácilmente serializables.
Se generaron las clases DTO con los atributos necesarios para poder copiar la información y poder lograr una exitosa serialización.