**Informe Trabajo Práctico**

**Programación III**

Integrantes:

Martin Velazquez (legajo: 39963713/2015

Facundo Cardenas (legajo: 42054625/2016)

Rodrigo Paz (legajo: 36788839/2014)

Profesores:

● Patricia Bagnes - pbagnes@ungs.edu.ar

● Javier Marenco - [jmarenco@ungs.edu.ar](mailto:jmarenco@ungs.edu.ar)

**Trabajo Práctico *“Equipo de Trabajo”***

I**ntroducción y desarrollo del proyecto:**

El trabajo propuesto por la materia consta en hacer una aplicación que genere el equipo de trabajo, en nuestro caso de Desarrollo, con sus roles y condiciones previamente enunciadas con Algoritmos de BackTracking/Fuerza Bruta para poder aplicar la decisión de armar el equipo más Optimo , teniendo en cuenta la cantidad mínima y máxima que es condicionada en la presentación del enunciado del Trabajo Practico, armando un equipo de trabajo del área de Desarrollo de Software lo más eficiente posible, respetando los patrones de diseño y soluciones utilizados anteriormente donde se gestiona el funcionamiento de la lógica con la interfaz y testing separado. En el transcurso de este Proyecto se explora diferentes utilidades variadas en las cuales se aplican métodos de diseño, de implementación y de estructura de datos que facilitan la tarea para el desarrollo del código y comprensión de los Items asignados ya que la solución no suele ser la mejor ya que es exponencial pero el algoritmo es exacto.

**Detalle sobre el funcionamiento del código:**

***Clases correspondientes al paquete de Interfaz/Controlador:***

Clases de Interfaz: Estas clases están compuestas sobre lo que se va a reflejar en la interfaz de la aplicación y parte de lo que vamos a utilizar de la clase de la lógica de la aplicación que gestiona los empleados para que se visualice en pantalla, es decir, las clases de interfaz a su vez es el controlador a nivel diseño.

**InterfazBotones**: Clase encarga de gestionar la acción de los botones que se visualizan por la interfaz.

**Métodos:**

**Buffer():** almacena los botones en la aplicaciones.

**AñadirQuitarIncompatibilidad()**: método encargado de gestionar la acción del botón de QuitarIncompatibilidad

**AñadirQuitarEmpleado()**: Gestiona la acción del botón que quita al empleado.

**accionFuerzaBruta()**: botón encargado de gestionar la acción de algoritmo que genera el grupo de trabajo óptimo.

**accionesBotones()**: Gestiona la acción de los botones restantes de la aplicación.

**InterfazChoice:** Clase encargada de actualizar los Choices (Opciones) en la interfaz a la hora de realizar una acción sobre un empleado.

**Metodos:**

**Buffer():** Almacena los datos que se pueden elegir como opciones en la aplicación.

**bufferChoiceRol():** Almacena las opciones de los roles que existen.

**actualizarListas():** Actualiza la lista de empleados luego de comprobar que hubo un evento en la aplicación.

**IntefazPrograma:** Clase encargada de ejecutar la aplicación**.** Solo se encarga de inicializar la InterfazGrafica y hacer visible el frame.

**InterfazTexto:** Clase encargada de gestionar las cajas de textos en la Interfaz.

**Metodos**:

**Buffer():** Agrega las cajas de textos en la aplicación.

**getText():** Obtiene el texto de la caja según el int de parámetro.

**setValue(),getValue():** Retorna el texto según el int que es pasado por parametro.

**InterfazTextoaddEmpleados:** Clase de la interfaz encargada de almacenar los textos que determinan la acción para los empleados.

**Metodos:**

**Buffer():** Almacena los textos los cuales reflejan donde tiene que agregar la informacion el usuario en la aplicación.

**getNombre(),getDNI(),getApellido():** Obtiene el Dni, nombre y apellido del empleado que ingresa en la aplicación.

**Actualizar ():** actualiza el DNI, Apellido y Nombre de la aplicación dejándolo vacío.

**InterfazTextoRolesEmpleados:** Clase de la interfaz que almacena los textos de cada Rol a elegir por Empleado.

**Métodos:**

**Buffer():** Agrega los roles del empleado a elegir en la aplicación.

**InterfazGrafica:** Clase encargada de gestionar toda la información de la interfaz a la aplicación.

**Métodos:**

**Buffer():** almacena todas las instancias de la interfaz en la aplicación, es decir, instancias de demás clases para que se inicialicen las acciones de la interfaz y se pueda visualizar la aplicación.

**InterfazListaEmpleados:** Clase encargada de gestionar las listas de los roles que podemos visualizar en la aplicación.

**Metodos:**

**Buffer():** Almacena la lista de empleados por diferentes listas con el rol correspondiente por cada una de ellas.

**actualizarListas():** actualiza las listas de los empleados con el rol correspondiente en cada una de ellas según la acción que se realice.

***Clases correspondiente al Código de Negocio/Producción que están contenidas en el paquete de CodigoNegocio:* Estas clases definen el comportamiento lógico de la aplicación, una clase que gestiona sus equipos y algoritmos de fuerza bruta que se aplica para tener el equipo óptimo.**

**Empleado:** Clase que contiene la información del empleado y su comportamiento.

**Metodos:**

**getRol(),getDni(),getNombre(),getApellido():** Obtiene información del empleado.

**Equals():** Compara dos empleados si son iguales, con la información previa anteriormente brindada.

**toString():** Muestra el empleado con toda su información.

**EquipoTrabajo:** Clase encargada de gestionar acciones sobre los empleados del grupo de trabajo.

**Metodos:**

**crearGrupoTrabajo():** Crea el grupo de trabajo mediante algoritmos de fuerza bruta

**getLider(),getArquitecto(),getProgramador(),getTester():** Obtiene la lista de los empleados, según cada rol.

**getCantIncompatible():** Devuelve cantidad de empleados incompatibles de un empleado.

**getEmpleados():** Obtengo los empleados del equipo de trabajo.

**perteneceEmpleado():** Se consulta si el empelado pertenece al equipo de trabajo por DNI.

**ordenarEmpleados():** es un método que se sobrecarga en otro con el mismo nombre que ordena cada lista de empleados según su Rol comparando Incompatibilidades.

**nuevoEmpleado():** Agrega un nuevo Empleado al equipo de trabajo.

**addIncompatibilidad**(): Agrega incompatibilidad entre dos empleados

**quitarIncompatibilidad():** Quita incompatibilidad entre dos empleados.

**esCompatibleConGrupoTrabajo():** Consulta si un empleado es compatible al resto del equipo de trabajo

**clone():** genera un clon del EquipoTrabajo

**toString():** Muestra el equipo de trabajo.

**FuerzaBruta:** Clase encarga de ejecutar el método crearGrupo() que crea el algoritmo óptimo para crear el grupo de trabajo.

**Metodos:**

**crearGrupo():** Método encargado de aplicar el algoritmo de fuerza bruta que gestiona la clase de EquipoTrabajo.

**getEquipo():** Obtiene el equipo auxiliar nuevo de la consecuencia de haber aplicado fuerza bruta.

**Run():** ejecuta el Thread donde va a estar el hilo en paralelo para la fuerza bruta.

***Clase correspondiente al Testing Unitario Obligatorio (Clase: TestProduccion):*** Esta clase se encarga de testear las clases y/o métodos que se encuentran en el código de producción para el funcionamiento de la aplicación realizando casos de uso.

**TestProduccion:**

**Metodos:**

***EmpleadosIgualesTest():*** Verifica si dos empleados son iguales.

***EmpleadosDistintosTest():*** Verifica si dos empleados son distintos.

***AgregarEmpleadoTest():*** Agrega un empleado al equipo de trabajo y verifica si se encuentra en el mismo.

***QuitarEmpleadoTest():*** Quita un empleado del equipo de trabajo y verifica que no exista.

***esCompatibleTest():*** Verifica si un empelado es compatible con el equipo de trabajo.

***cantidadEmpleadosTest():*** Verifica que la cantidad de empleados sea la correcta.

***quitarEmpleadoListTest():*** Quita un empleado de la lista del rol de ese empleado.

**Problemática y Conclusiones Finales:**

En el trabajo propuesto tuvimos algunos inconvenientes , siendo uno de los que contenía más dificultad el hecho de trabajar sobre un algoritmo exponencial que es exacto y encontrar una solución, que no es del todo optima al ser un problema NP.

Nos ayudó a comprender estos problemas que no tienen una resolución polinomial, como otros tantos algoritmos. También pudimos diseñar soluciones acordes al patrón de diseño, división de responsabilidades y aplicar las lecturas vistas en el aula para mejorar la calidad del código y hacer Test eficientes.