GlassWindow

Enunciado número 13

Informe del Proyecto Final de las asignaturas: Diseño y Programación Orientado a Objetos y Diseño de Interfaces y Pruebas

Vicente Samuel Garófalo Jerez

Carlos Daniel Vilaseca Illnait

La Habana, 2022

### **Resumen**

Este proyecto representa el desarrollo de una aplicación de escritorio utilizada para gestionar la bolsa de ubicación laboral, o sea, gestionar las ofertas y demandas de empleos de varias empresas y los candidatos. La aplicación está diseñada como una solución multiplataforma, con una interfaz consistente y una arquitectura modular y desacoplada.

### **Índice**

[**Resumen** 2](#_Toc107151083)

[**Índice** 3](#_Toc107151084)

[**Introducción** 1](#_Toc107151085)

[Planificación de Actividades 2](#_Toc107151086)

[**Desarrollo** 3](#_Toc107151087)

[Tarjetas CRC 3](#_Toc107151088)

[Diagramas de clases UML 5](#_Toc107151089)

[Propuestas de reportes 5](#_Toc107151090)

[Patrones de diseño 6](#_Toc107151091)

[Singleton 6](#_Toc107151092)

[MVC (Model View Controller) 7](#_Toc107151093)

[Validaciones 7](#_Toc107151094)

[Vistas y componentes 9](#_Toc107151095)

[Patrones de diseño de interfaces 17](#_Toc107151096)

[*CalendarStrip* 17](#_Toc107151097)

[*Fill-in-the-blanks* 17](#_Toc107151098)

[*Search Box* 17](#_Toc107151099)

[Pruebas realizadas 18](#_Toc107151100)

[Pruebas de caja blanca *(white box testing)* 19](#_Toc107151101)

[Pruebas de caja negra *(black box testing)* 24](#_Toc107151102)

[Pruebas unitarias con Junit 24](#_Toc107151103)

[**Conclusiones** 26](#_Toc107151104)

[**Recomendaciones** 27](#_Toc107151105)

[**Bibliografía** 28](#_Toc107151106)

### **Introducción**

En una agencia empleadora se desea gestionar la bolsa de ubicación laboral. Dicha agencia se encarga de gestionar las ofertas y demandas de empleo de variadas empresas. De cada empresa se conoce: nombre, dirección, teléfono y sector en el que se encuentra ubicada (turismo, salud, educación, entre otros). Una empresa puede realizar solicitudes para tantos puestos de trabajo como necesite, así como solicitar la eliminación de puestos de trabajo que solicitó. De cada oferta se conoce: ID, rama a la que pertenece, salario del puesto y empresa a la que pertenece esta oferta. Al llegar un candidato a empleado a la agencia se le recogen los siguientes datos: carné de identidad, nombre, sexo, dirección, teléfono, nivel de escolaridad, especialidad, rama en la que desea trabajar (directivo, ingeniero, recursos humanos, entre otras) y años de experiencia en su área de especialidad. En dependencia de la rama en la que desee trabajar, se deben incluir otros datos (un candidato que desee trabajar en turismo debe presentar un certificado de idioma, un candidato que desee incorporarse a la plantilla de custodios debe presentar sus calificaciones en las pruebas de eficiencia física, así como su número de historia clínica). Una vez ingresado los datos al sistema, se comprueba si existen actualmente ofertas de empleo de la rama que le interesa. Se debe crear una funcionalidad en cada candidato, en la cual, dada una oferta, se sepa si el candidato puede o no aspirar a ese empleo. De existir alguna oferta de empleo para el candidato, se programa una cita con la empresa interesada. El candidato puede asistir a todas las entrevistas que desee, pero solo puede asistir a una cita al día. De no existir un empleo que cumpla con los requerimientos del candidato, sus datos quedan almacenados en el sistema. Cuando una empresa realice una solicitud de empleo, los candidatos inscritos en el sistema deben ser revisados a ver si existe alguno con las cualidades necesarias para satisfacer dicha solicitud. La agencia empleadora cuenta con un registro mensual de la planificación de entrevista, donde se tiene por día, para cada empleo que solicita una empresa, si hay planificada una entrevista o no.

El objetivo de este proyecto es crear una aplicación de escritorio multiplataforma para gestionar las ofertas y demandas de empleo de varias empresas.

#### Planificación de Actividades

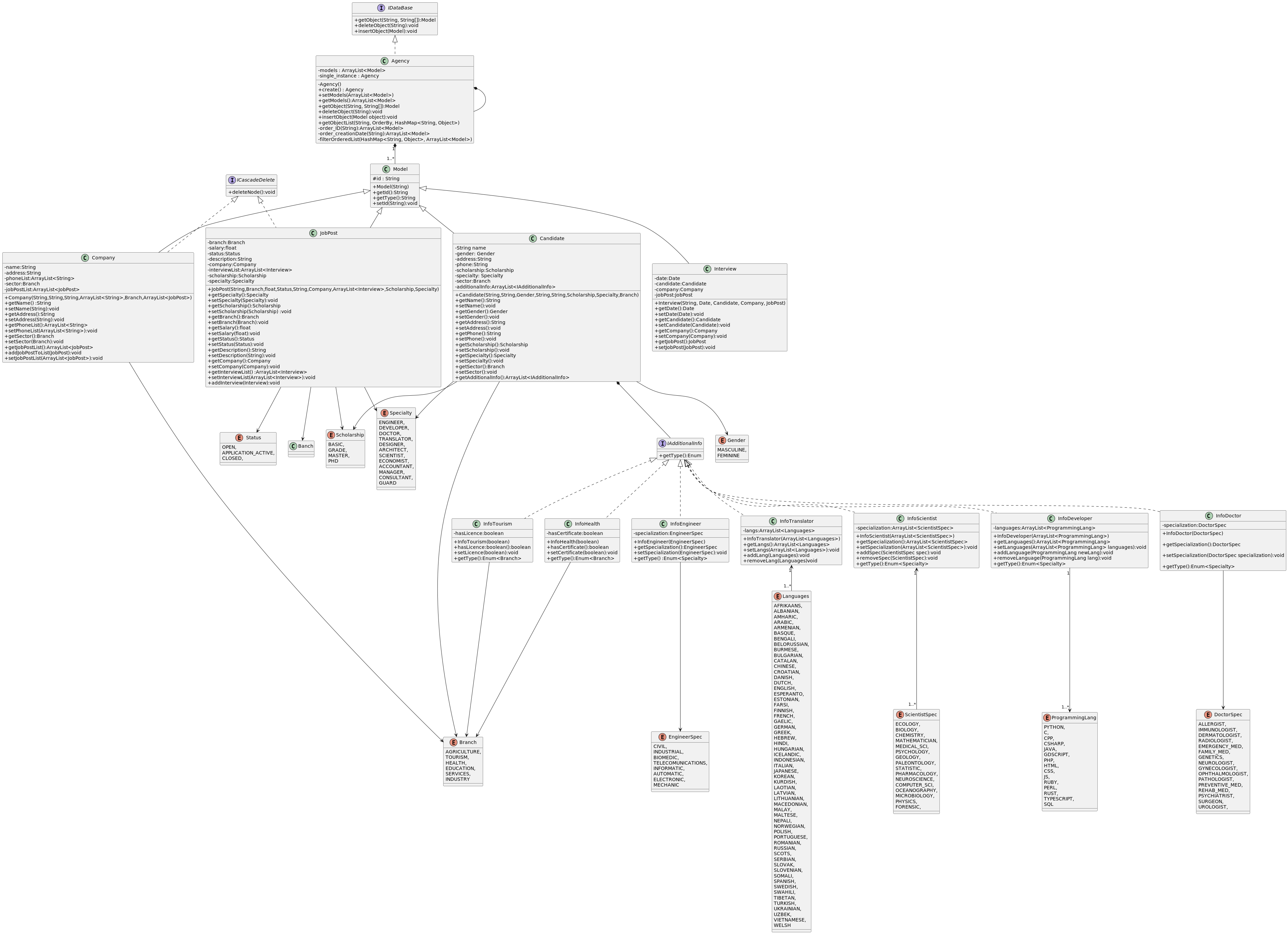
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tareas | Diseñar UML | Elaborar UML | Elaborar CRC | Proponer Reportes | Elaborar Informe del Corte 1 |
| Responsable | Vicente | Carlos Daniel | Carlos Daniel | Vicente y Carlos | Vicente |
| Límite | 2/5/2022 | 9/5/2022 | 13/5/2022 | 12/5/2022 | 13/5/2022 |

### **Desarrollo**

#### Tarjetas CRC

|  |  |
| --- | --- |
| **Candidate** |  |
| **Responsabilidades** | **Colaboradores** |
| Decir su nombre |  |
| Decir su nivel de escolaridad |  |
| Decir la rama en la que desea trabajar |  |
| Decir su carné de identidad |  |
| Decir su dirección |  |
| Decir su teléfono |  |
| Decir su sexo |  |
| Decir su especialidad |  |
| Decir los años de experiencia en su área de especialidad |  |
| Decir dada una oferta de trabajo si puede aplicar o no | JobPost |
|  | |
| **Company** |  |
| **Responsabilidades** | **Colaboradores** |
| Solicitar creación de ofertas de trabajo | Agency |
| Solicitar eliminación de ofertas de trabajo | Agency |
| Decir nombre |  |
| Decir su dirección |  |
| Decir su sector |  |
| Decir su teléfono |  |
| Almacenar los datos de las ofertas de trabajo creadas | JobPost |
|  | |
| **JobPost** |  |
| **Responsabilidades** | **Colaboradores** |
| Decir su ID |  |
| Decir su rama |  |
| Decir su estado |  |
| Decir el salario del puesto |  |
| Decir empresa a la que pertenece |  |
| Decir descripción del puesto |  |
|  | |
| **Agency** |  |
| **Responsabilidades** | **Colaboradores** |
| Decir dada una oferta de trabajo si un candidato puede aplicar | JobPost, Candidate |
| Programar las citas entre empresas y candidatos | JobPost, Candidate, Company, Interview |
| Eliminar las ofertas de trabajo | JobPost |
| Crear las ofertas de trabajo solicitadas | Company, JobPost |
| Verificar dada una fecha si un candidato puede asistir a una entrevista | Interview, Candidate |
| Dada una empresa decir sus ofertas de trabajo y los candidatos que han aplicado a los mismos en el mes especificado | Company, JobPost, Candidate |
| Dado un día del mes, decir las entrevistas planificadas para ese día | Interview |
| Almacenar los datos de las empresas a las que se les gestiona la bolsa de empleo | Company |
| Almacenar los datos de todos los candidatos registrados | Candidate |
| Almacenar los datos de todas las ofertas de trabajo creadas | JobPost |
| Almacenar los datos de todas las entrevistas planificadas | Interview |
|  |  |
| **Interview** |  |
| **Responsabilidades** | **Colaboradores** |
| Decir la información del candidato | Candidate |
| Decir la información de la oferta de trabajo | JobPost |
| Decir la fecha en la que está planificada |  |
| Solicitar la verificación de la fecha de la entrevista | Agency, Candidate |
| Almacenar los datos del candidato que atiende a la entrevista | Candidate |
| Almacenar la fecha y hora de la entrevista |  |
| Almacenar los datos de la oferta de trabajo | JobPost |

#### Diagramas de clases UML



#### Propuestas de reportes

1. Para los candidatos mostrar las ofertas de trabajo que tiene disponibles de acuerdo con sus calificaciones y la rama en la que desea trabajar. Este reporte es importante para servir en la interfaz del candidato solo las ofertas de trabajo que son relevantes para él. En este reporte hay que consultar información de la clase Candidate y usarla para filtrar la lista de JobPost que se encuentra compuesta en la clase Agency.
2. Para una compañía especifica, obtener los nombres de todos los candidatos que han aplicado a sus ofertas de trabajo, con la opción de limitar dicha lista de nombres a los que han aplicado en un mes específico. Vista importante para que las compañías tengan una vista general sobre los candidatos y puedan acceder a rápidamente a sus perfiles para ver sus cualificaciones. Este reporte debe escanear toda la lista de JobPost, para obtener de cada una la lista de Interview, filtrar esa lista por la fecha, y de cada entrevista obtener el Candidate.
3. Tanto para una empresa o un empleado, un reporte que muestre las entrevistas planificadas por cada día del mes. Reporte útil para poblar la vista de calendario. Sería una lista de listas con la siguiente estructura {[Interview1, Interview2, Interview3], [Interview4, Interview5]} donde la posición de índice 0 corresponde al día 1 del mes, y el índice 1 corresponde al día 2.
4. Un reporte que muestre dada una compañía, todas las entrevistas planificadas, agrupadas por oferta de trabajo a la que corresponden. El resultado sería una lista de listas con la siguiente estructura {[Interview1, Interview2], [Interview3], [Interview4]} donde la posición de índice 0 sería la primera JobPost publicada por la empresa dada y así sucesivamente.

#### Patrones de diseño

##### Singleton

El patrón de diseño Singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

En este caso el singleton es la Agencia que gestiona la bolsa de empleo.

##### MVC (Model View Controller)

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

* El Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
* La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
* El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

#### Validaciones

Main view (*vista principal*)

* **Search:** sin validación (es un campo opcional).

New Company view (*vista nueva compañía*)

* **Description**: sin validación (campo opcional)
* **Address**: sin validación (campo opcional)
* **Phone**: en caso de usarse, debe ser de 8 dígitos, se hace uso de la excepción *InvalidPhoneException* en caso de que no cumpla con la validación. Es un campo opcional.
* **Sector**: sin validación (es una lista desplegable)

New Candidate view (*vista nuevo candidato*)

* **Candidate ID:** debe ser de 11 dígitos, el tercer y cuarto dígitos que conforman el mes de nacimiento debe ser mayor que 01 y menor que 12, en cuanto al quinto y sexto dígitos que conforman el día del mes de nacimiento, si el mes es 01,03,05,07,08,10 o 12, entonces el día se debe encontrar entre 01 y 31, si el mes es 04,06,09 u 11, entonces el día se debe encontrar entre 01 y 30, en el caso de que sea 02 el mes se validó que el día se encuentre entre 01 y 29, dado que no se puede validar si un año es bisiesto o no con tan solo las dos cifras finales del año, lo mismo ocurre con la cifra 7 del ID, correspondiente al siglo de nacimiento: si nació en el siglo XIX, esté dígito será 9, si el siglo es XX el dígito estará entre 0 y 5, y si es siglo XXI el dígito estará entre 6 y 8, pero de nuevo, no hay como definir el siglo con solo dos cifras del año. Se hace uso de la excepción *InvalidIDException* en caso de que no se cumpla con las validaciones anteriores. En caso de que ya exista un candidato registrado con ese ID se lanza la excepción *DuplicatedIDException*.
* **Full Name:** No debe contener solo espacios, o dejarse el campo vacío, para esta validación se utilizó el método. *isBlank()* que retorna *true* en estos casos, tampoco debe contener números, en caso de ser necesario se deben usar números romanos. Se hace uso de la excepción *InvalidNameException* en caso de que no se cumplan las validaciones.
* **Address:** sin validación (campo opcional).
* **Phone:** en caso de usarse, debe ser de 8 dígitos, se hace uso de la excepción *InvalidPhoneException* en caso de que no cumpla con la validación. Es un campo opcional.
* **Years of Experience:** debe ser un numero mayor que 0, se usa la excepción *InvalidYearsOfExp* en caso de no cumplirse esta validación.
* **Additional Info:** Campo obligatorio en dependencia de la rama y especialidad.

New JobPost view (*vista nueva oferta de trabajo)*

* **Company:** Debe ser una compañía ya registrada, en caso de que no se usa la excepción *ModelNotFoundException*.
* **Description:** sin validación (campo opcional).
* **Salary:** Debe ser un número, mayor que 0, en caso de que no, se hace uso de la excepción *InvalidSalaryException*.
* **Branch:** sin validación (campo opcional).
* **Specialty:** sin validación (campo opcional).

New Company view (*vista nueva compañía*)

* **Description:** sin validación (campo opcional).
* **Address:** sin validación (campo opcional).
* **Phone:** en caso de usarse, debe ser de 8 dígitos, se hace uso de la excepción *InvalidPhoneException* en caso de que no cumpla con la validación. Es un campo opcional.
* **Sector:** sin validación (es una lista desplegable).

Arrange Interview view (*vista creación de entrevista*)

* **Date:** debe ser una fecha posterior a la fecha actual, en caso de que no se hace uso de la excepción *InvalidDateException*.
* **Time:**
  + **Hour**: debe ser un numero entre 01 y 12.
  + **Minute**: debe ser un numero entre 00 y 59.
* **Candidate**: sin validación (es una lista desplegable).

\*Para las vistas de edición de estos modelos se tienen en cuenta las mismas validaciones.

#### Vistas y componentes

En todas las vistas se hace uso de un color neutral grisáceo, a excepción de las ventanas de mensajes de error que se usa un color de fuente rojo para dar énfasis de que es un error que acaba de ocurrir. No se hace uso de gráficos ni iconografía. Se utiliza la fuente de texto por defecto del sistema que use la aplicación.

A continuación, las propuestas de vistas con imágenes de referencia:

Graphical user interface, text

Description automatically generated

(img. 1) Vista principal

La **vista principal** (véase img.1) está compuesta de la siguiente manera:

* En la parte superior de la vista se encuentra un *MenuBar* que contiene a los componentes *Menu* para: *File*, *Edit* y *Help,* estos a su vez contienen sus correspondientes *MenuItem,* como por ejemplo en el *Menu File* se encuentran los *MenuItem: Open Recent, Preferences, Quit*, entre otros, pero en el *Menu File* hay también otro *Menu*, llamado *New*, que contiene precisamente tres *MenuItem*: *Candidate, Company y JobPost,* que son los que en correspondencia se utilizarán para la adición de nuevos candidatos, compañías, y ofertas de trabajo al sistema a través de las vistas de *New Candidate, New Company* y *New JobPost,*  respectivamente.
* En el cuerpo de la ventana se hace uso de un *SplitPanel* para dividir el menú lateral y el panel correspondiente a donde se mostrarán los datos, o panel de datos para abreviar.
* En el panel de la izquierda tenemos el menú lateral con tres componentes *Button*, para *Companies, Candidates* y *JobPosts*, a través de los cuales se cambia la información que se muestra en el panel de datos en correspondencia del botón que se presione.
* En el panel de datos tenemos, en la parte superior, una barra de búsqueda compuesta por un *TextField* y un *Button*; justo debajo de esta barra de búsqueda tenemos, dentro de un *ScrollPane*, un componente *ListView*, donde se listan los datos en correspondencia del botón del panel de la izquierda que se presione.

A screenshot of a computer

Description automatically generated with medium confidence

(Img. 2) Formulario para la creación/edición de JobPosts

La **vista para la creación/edición de ofertas de trabajo** (véase img. 2) está compuesta de la siguiente manera:

* En la parte superior tenemos un componente *Text* para el encabezado.
* Luego tenemos varios componentes *Label* a los que se le corresponden diferentes componentes. Al *Label* *Company* le corresponde un *TextField*, a *Description* le corresponde un componente *TextArea*, a *Salary* un componente *Spinner* y junto a este un componente *Text* con el símbolo “$”, y a los *Label Branch* y *Specialty* le corresponde a cada uno un componente *ComboBox*.
* Al final del formulario se encuentran los componentes *Button* para guardar y cancelar respectivamente.

Graphical user interface, application, Word

Description automatically generated

(Img. 3) Formulario para la creación/edición de Candidates

La **vista para la creación/edición de Candidatos** (véase img. 3) está compuesta de la siguiente manera:

* En la parte superior tenemos un componente *Text* para el encabezado. En el caso de edición este componente tendrá el texto correspondiente al nombre del candidato a editar, en el caso de creación el texto será “New Candidate”.
* Luego tenemos varios componentes *Label* a los que se le corresponden diferentes componentes.
  + *Candidate ID, Full Name, Address, Phone 🡺 TextField*
  + *Gender, Scholarship, Specialty, Branch 🡺 ComboBox*
  + *Years of experience 🡺 Spinner*
  + *Other Data 🡺*
* Al final del formulario se encuentran los componentes *Button* para guardar y cancelar respectivamente.

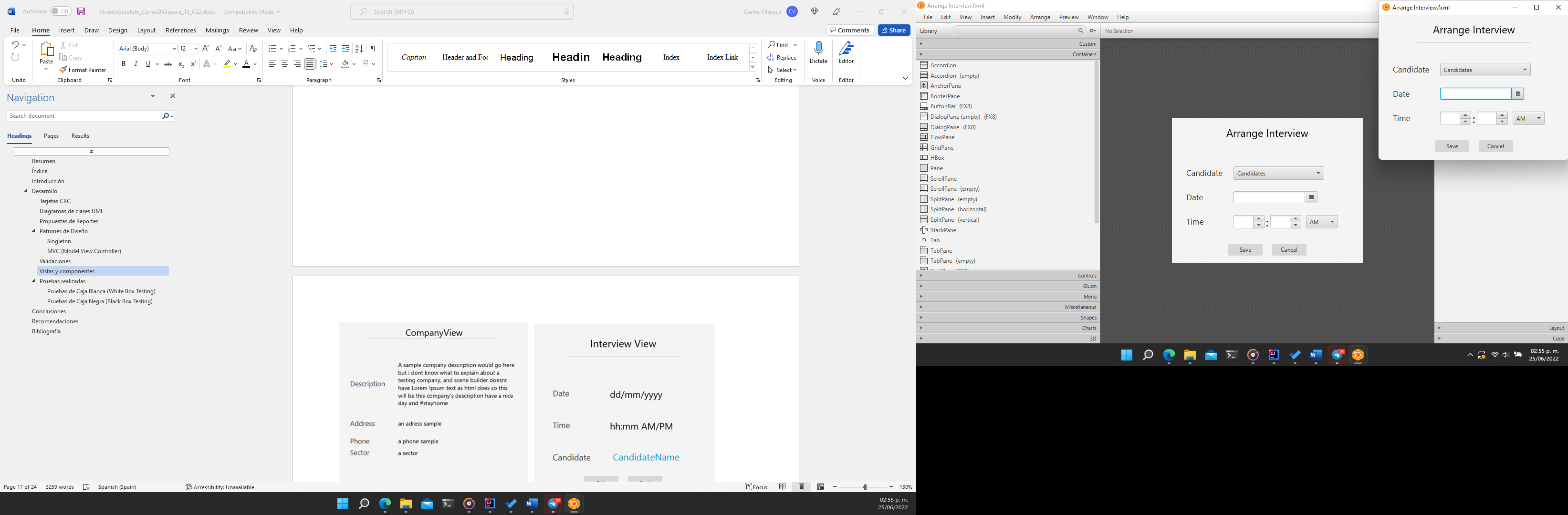
A screenshot of a computer

Description automatically generated with medium confidence

(Img. 4) Prototipo de Formulario para la creación/edición de Companies

La **vista para la creación/edición de Compañías** (véase img. 4) está compuesta de la siguiente manera:

* En el caso de edición este componente tendrá el texto correspondiente al nombre de la compañía a editar, en el caso de creación el texto será “New Company”.
* Luego tenemos varios componentes *Label* a los que se le corresponden diferentes componentes:
  + *Name, Address, Phone 🡺 TextField*
  + *Description 🡺 TextArea*
  + *Sector 🡺 ComboBox*
* Al final del formulario se encuentran los componentes *Button* para guardar y cancelar respectivamente



(Img. 5) Formulario para la creación/edición de Interviews.

La **vista para la Programación de Entrevistas** (véase img. 5) está compuesta de la siguiente manera:

* En la parte superior tenemos un componente *Text* para el encabezado.
* Luego tenemos varios componentes *Label* a los que se le corresponden diferentes componentes:
  + *Date 🡺 DatePicker*
  + *Time 🡺 Spinner x2, ComboBox*
  + *Candidate 🡺 ComboBox*
* Al final del formulario se encuentran los componentes *Button* para guardar y cancelar respectivamente.

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

(Img. 6) Vistas para la consulta de datos específicos

Las **vistas para la consulta de datos específicos** (véase img. 6) ya sea de una compañía, un candidato, una oferta de trabajo o una entrevistaestá compuesta de la siguiente manera:

* En la parte superior tenemos un componente *Text* para el encabezado, en el caso de la compañía y el candidato el texto es el nombre correspondiente.
* Luego tenemos varios componentes *Label* a los que se le corresponden otros componentes *Text*.
* En la parte inferior de cada vista se encuentran los componentes *Button*, *Edit* y *Back*, a través de los cuales se accede a las correspondientes vistas de edición, y se regresa a la vista principal respectivamente, en el caso de la vista de oferta de trabajo también hay un botón de *Apply*, que lleva a la vista de la img. 5 para programar la entrevista en caso de que los datos del candidato sean los requeridos por la oferta de trabajo.

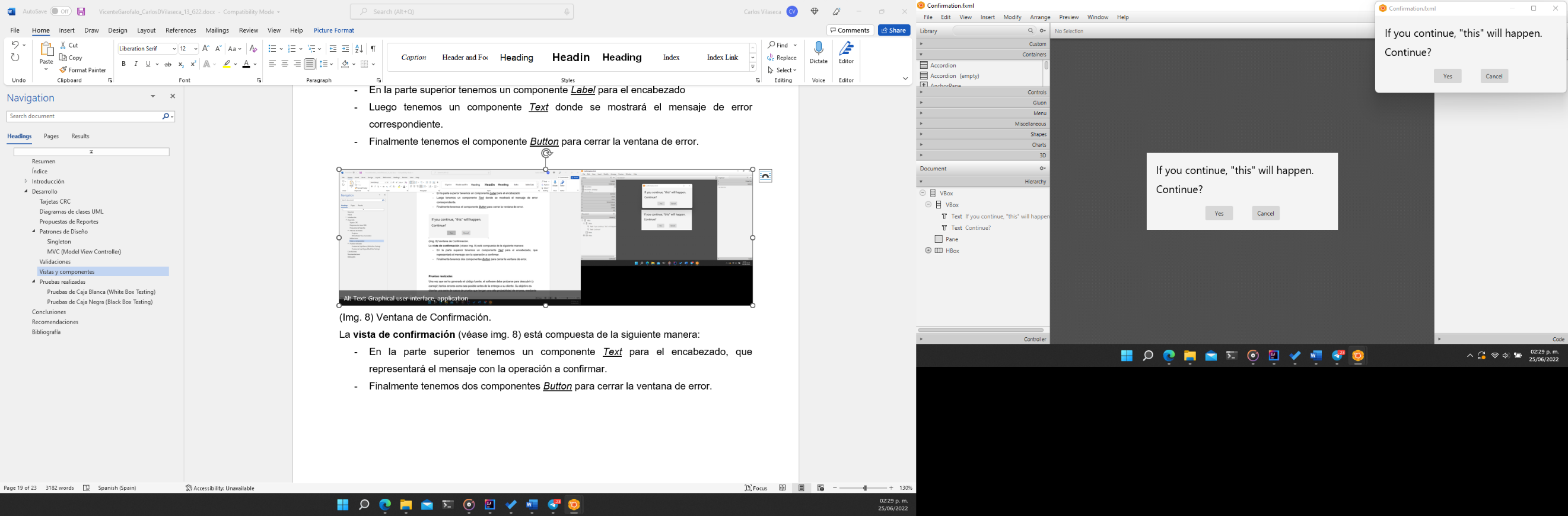
Graphical user interface, application

Description automatically generated

(Img. 7) Ventana de Mensaje de error

La **vista de mensaje de error** (véase img. 7)está compuesta de la siguiente manera:

* En la parte superior tenemos un componente *Label* para el encabezado
* Luego tenemos un componente *Text* donde se mostrará el mensaje de error correspondiente.
* Finalmente tenemos el componente *Button* para cerrar la ventana de error.



(Img. 8) Ventana de Confirmación.

La **vista de confirmación** (véase img. 8)está compuesta de la siguiente manera:

* En la parte superior tenemos un componente *Text* para el encabezado, que representará el mensaje con la operación a confirmar.
* Finalmente tenemos dos componentes *Button* para confirmar la operación y cancelarla.

#### Patrones de diseño de interfaces

Las interfaces gráficas de usuario se han vuelto tan comunes que ha surgido una amplia variedad de patrones de diseño de interfaces de usuario. Un patrón de diseño es una abstracción que prescribe una solución de diseño a un problema de diseño específico y bien acotado. A continuación, se detallan algunos de los patrones de diseño utilizados en la aplicación.

##### *CalendarStrip*

En esta aplicación se emplea el patrón de diseño ***CalendarStrip*** (véase img. 5 e img. 9) que produce un calendario continuo y desplazable en el que se resalta la fecha actual y se pueden seleccionar fechas futuras seleccionándolas del calendario.

A screenshot of a computer

Description automatically generated with medium confidence

Img. 9 Patrón de diseño *CalendarStrip*.

##### *Fill-in-the-blanks*

Se emplea el patrón ***fill-in-the-blanks*** en todas las vistas de creación/edición (véase img. 1-4)

Este patrón consiste en permitir la entrada de datos en componentes *Text Box o Text Field,* incluyendo al final un componente *Button* como *“Go”, “Submit”, “Next”,* entre otros posibles el cual indica el inicio de la validación de los datos entrados.

##### *Search Box*

Se emplea el patrón ***Search Box*** en la vista principal (véase img. 1). Este patrón de diseño permite al usuario la búsqueda de datos específicos, en el caso de esta aplicación permite en dependencia del panel de datos que se encuentre (Companies, Job Posts, Candidates) la búsqueda del modelo que desee.

#### Pruebas realizadas

Una vez que se ha generado el código fuente, el software debe probarse para descubrir (y corregir) tantos errores como sea posible antes de la entrega a su cliente. Su objetivo es diseñar una serie de casos de prueba que tengan una alta probabilidad de errores, mediante las técnicas de prueba de software. Estas técnicas proporcionan una guía sistemática para diseñar pruebas que:

1. Ejercitan la lógica interna y las interfaces de cada componente de software.
2. Ejercitan los dominios de entrada y salida del programa para descubrir errores en la función, el comportamiento y el rendimiento del programa.

Para aplicaciones convencionales, el software se prueba desde dos perspectivas diferentes:

1. La lógica interna del programa se ejerce utilizando técnicas de diseño de casos de prueba de "caja blanca".
2. los requisitos de software se ejercen utilizando técnicas de diseño de casos de prueba de "caja negra".

Los casos de uso ayudan en el diseño de pruebas para descubrir errores a nivel de validación de software. En todos los casos, la intención es encontrar el máximo número de errores con la mínima cantidad de esfuerzo y tiempo.

##### Pruebas de caja blanca *(white box testing)*

public Model getModelWithID(String id) throws IdNotFoundException {  
 ArrayList<Model> models = getModels();  
 for(int i = 0; i < models.size(); i++)  
 if(models.get(i).getId().equals(id))  
 return models.get(i);  
 throw new IdNotFoundException(id);  
}

|  |  |
| --- | --- |
|  | Nodos |
| 1 | public Model getModelWithID(String id) throws IdNotFoundException |
| 2 | ArrayList<Model> models = getModels(); |
| 3 | int i = 0 |
| 4 | i < models.size() |
| 5 | i++ |
| 6 | models.get(i).getId().equals(id) |
| 7 | return models.get(i) |
| 8 | throw new IdNotFoundException(id) |

Graf. 1

CP

1. 1-2-3-4-8 (models está vacía)
2. 1-2-3-4-6-7 (el modelo con el id necesitado esta en la primera posición de la lista)
3. 1-2-3-4-6-5-4-7 (el modelo con el id necesitado se encuentra en alguna posición de la lista)
4. 1-2-3-4-6-5-4-8 (el modelo con el id necesitado no se encuentra en la lista)

*\*Para estos casos de prueba solo interesa el atributo ID de los objetos Model, así que en lugar de representar el objeto Model con todos sus atributos, lo representamos como el ID.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CASO DE PRUEBA** | | | | | |
| **Clase** | Agency | | | | |
| **Método** | getModelWithID | | | | |
| **Desarrollador** | Carlos Daniel Vilaseca | | | | |
| **Probador** |  | | | | |
| **Fecha** |  | | | | |
| **Combinaciones de valores de entrada** | | | **Técnica de diseño empleada** | **Resultados Esperados** | **Resultados reales** |
| **CP** | **Nombres de Variables de entrada (V1, V2, …, Vn)** | **Para cada variable de entrada:**  **variable=valor** |
| 1 | Id,  models | “01060568481”  [] | Caja blanca-Camino simple | Se lanza la excepción IdNotFoundException y se muestra la ventana de error |  |
| 2 | Id,  models | “01060568481”  [“01060568481”,”12121299999”] |  | Se retorna el modelo con el id “01060568481” |  |
| 3 | Id  models | “01060568481”  [“15120512121”, “01060568481”, ”12121299999”] |  | Se retorna el modelo con el id “01060568481” |  |
| 4 | Id  models | “01010100000”  [“15120512121”, “01060568481”, ”12121299999”] |  | Se lanza la excepción IdNotFoundException y se muestra la ventana de error |  |

Tabla 1. Casos de Prueba del método getModelWithID() de la clase Agency.

public ArrayList<Candidate> getAppliances (String company, int month){  
 ArrayList<Candidate> result = new ArrayList<>();  
 Company c = (Company)getModelWithID(company);  
 ArrayList<String> jobPosts = new ArrayList<>(c.getJobPostList());  
 ArrayList<String> interviews;  
 JobPost tempJP;  
 for (int j = 0; j < jobPosts.size(); j++) {  
 tempJP = (JobPost)getModelWithID(jobPosts.get(j));  
 interviews = tempJP.getInterviewList();  
 for (int i = 0; i < interviews.size(); i ++) {  
 Interview = (Interview) getModelWithID(interviews.get(i));  
 if (interview.getDate().getMonth() == month) {  
 Candidate = (Candidate)getModelWithID(interview.getCandidate());  
 result.add(candidate);  
 }  
 }  
 }  
 return result;  
}

|  |  |
| --- | --- |
|  | Nodos |
| 1 | public ArrayList<Candidate> getAppliances (String company, int month) throws IdNotFoundException |
| 2 | ArrayList<Candidate> result = new ArrayList<>(); Company c = (Company)getModelWithID(company; ArrayList<String> jobPosts = new ArrayList<>(c.getJobPostList()); ArrayList<String> interviews; JobPost tempJP; |
| 3 | int j = 0 |
| 4 | j < jobPosts.size() |
| 5 | j++ |
| 6 | tempJP = (JobPost)getModelWithID(jobPosts.get(j)); interviews = tempJP.getInterviewList(); |
| 7 | int i = 0 |
| 8 | i < interviews.size() |
| 9 | i++ |
| 10 | Interview = (Interview) getModelWithID(interviews.get(i)) |
| 11 | interview.getDate().getMonth() == month |
| 12 | Candidate = (Candidate)getModelWithID(interview.getCandidate()); result.add(candidate); |
| 13 | return result; |

CP para cada bucle

1. Saltar el bucle por completo.

2. Solo un paso a través del bucle.

3. Dos pasadas a través del bucle.

4. **m** pasadas a través del bucle donde **m < n**.

5. **n-1**, **n**, **n+1** pasadas a través del bucle.

Técnica para bucles anidados

1. Comience en el bucle más interno. Establezca todos los demás bucles en valores mínimos. 2. Realice pruebas de bucle simples para el bucle más interno mientras mantiene los bucles externos en sus valores mínimos de parámetros de iteración (por ejemplo, contador de bucles). Agregue otras pruebas para valores fuera de rango o excluidos.

3. Trabaje hacia afuera, realizando pruebas para el siguiente bucle, pero manteniendo todos los demás bucles externos en valores mínimos y otros bucles anidados a valores "típicos".

4. Continúe hasta que se hayan probado todos los bucles.

***\*****En este caso no tiene sentido probar n-1, n, n+1 pasadas por el bucle dado que no hay un máximo de entrevistas u ofertas de trabajo establecidas.*

*Para los CP 1-4 se considera el bucle externo fijado en su valor mínimo jobPosts.size() = 1.*

*Para los CP 5-8 se considera el bucle interno fijado en si valor típico interviews.size() = 1. A excepción del CP 5, donde este valor es 0, dado que si jobPosts.size() es 0, entonces no hay entrevistas.*

***\*****Dado que de las Interview solo nos interesan el atributo* ***Date*** *y el atributo* ***Candidate****, la representamos solo por estos atributos*

***\*****Dado que de los JobPost solo nos interesa el método que retorna la lista de Interview del JobPost actual (****getInterviewList()****), lo representaremos con esa lista*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CASO DE PRUEBA** | | | | | |
| **Clase** | Agency | | | | |
| **Método** | getAppliances | | | | |
| **Desarrollador** | Vicente Samuel Garofalo | | | | |
| **Probador** |  | | | | |
| **Fecha** |  | | | | |
| **Combinaciones de valores de entrada** | | | **Técnica de diseño empleada** | **Resultados Esperados** | **Resultados reales** |
| **CP** | **Nombres de Variables de entrada (V1, V2, …, Vn)** | **Para cada variable de entrada:**  **variable=valor** |
| 1 | company,  month,  interviews.size(),  interviews,  jobPosts.size(),  jobPosts | “AMD”,  6,  0,  [],  1,  [JobPost1] | Caja blanca-Técnica de bucles | Se salta el bucle por completo.  Retorna una lista vacia [] |  |
| 2 | company,  month,  interviews.size(),  interviews,  jobPosts.size()  jobPosts | “AMD”,  9,  1,  [{5/9/2022, Candidate1}],  1,  [JobPost1] |  | Pasa por el bucle una vez, y retorna una lista [Candidate1] |  |
| 3 | company,  month,  interviews.size(),  interviews,  jobPosts.size()  jobPosts | “AMD”,  9,  2,  [{5/9/2022,Candidate1},  {10/9/2022,Candidate2}],  1,  [JobPost1] |  | Pasa por el bucle dos veces y retorna una lista [Candidate1, Candidate2] |  |
| 4 | company,  month,  interviews.size(),  interviews,  jobPosts.size()  jobPosts | “AMD”,  9,  3,  [{5/8/2022,Candidate1},  {10/9/2022,Candidate2},  {11/9/2022,Candidate3}],  1,  [JobPost1] |  | Pasa por el bucle m veces, m < n, n=∞. m=3,  Retorna una lista [Candidate2, Candidate3] |  |
| 5 | company,  month,  interviews.size(),  interviews,  jobPosts.size()  jobPosts | “Last Quarter Studios”  7,  0,  [],  0,  [] |  | Se salta el bucle por completo.  Retorna una lista vacía [] |  |
| 6 | company,  month,  interviews.size(),  interviews,  jobPosts.size()  jobPosts | “Last Quarter Studios”  7,  1,  [{5/7/2022, Candidate1}],  1,  [JobPost1:  [{5/7/2022, Candidate1}]] | Técnica de bucles | Pasa por el bucle una vez,  retorna una lista [Candidate1] |  |
| 7 | company,  month,  interviews.size(),  interviews,  jobPosts.size(),  jobPosts | “Last Quarter Studios”  7,  1,  [{5/7/2022,Candidate1}],  [{5/9/2022,Candidate2}],  2,  [  JobPost1:  [{5/7/2022,Candidate1}],  JobPost2:  [{5/9/2022,Candidate2}]  ] | Técnica de bucles | Pasa por el bucle dos veces.  Retorna una lista [Candidate1] |  |
| 8 | company,  month,  interviews.size(),  interviews,  jobPosts.size()  jobPosts | “Last Quarter Studios”  7,  1,  [{5/7/2022,Candidate1}],  [{5/9/2022,Candidate2}],  [{6/7/2023,Candidate3}]  3,  [  JobPost1:  [{5/7/2022,Candidate1}],  JobPost2:  [{5/9/2022,Candidate2}],  JobPost3: [{6/7/2023,Candidate3}]  ] | Técnica de bucles | Pasa por el bucle m veces, m < n, n=∞, m=3.  retorna una lista [Candidate1, Candidate3] |  |

Tabla 2. Casos de Prueba del método getAppliances() de la clase Agency.

##### Pruebas de caja negra *(black box testing)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Identificador** | **Nombre HU** | | **Prioridad** | | **Puntos estimados** |
| HU1 | Encontrar Modelo | |  | |  |
| **Usuario:** | | | | | |
| **Descripción:** Se necesita encontrar un candidato, compañía, oferta de trabajo o entrevista, dado el ID | | | | | |
| **Programador Responsable:** Carlos Daniel Vilaseca | | | | | |
| **Escenarios** | | | | | |
| **Nombre** | **Condición** | | | | **Resultado esperado** |
| La lista de modelos del sistema está vacía | models.size() = 0 | | | | Se lanza la excepción IdNotFoundException y se muestra la ventana de error. |
| El modelo necesitado esta en la primera posición de la lista de modelos | i = 0  models.size() > 0  models.get(i).getId().equals(id) | | | | Se retorna el modelo con el ID deseado |
| El modelo con el id necesitado se encuentra en alguna posición de la lista | models.size() > 0  models.get(i).getId().equals(id) | | | | Se retorna el modelo con el ID deseado |
| El modelo con el id necesitado no se encuentra en la lista | models.size() > 0 | | | | Se Lanza la excepción IdNotFoundException. |
| **Variables** | | | | | |
| **Nombre** | | **Tipo o Dominio** | | **Descripción** | |
| id | | String | | Representa el identificador de un modelo ya sea Candidato, Compañía, Oferta de trabajo o Entrevista | |
| models | | Lista | | Lista con todos los modelos ya registrados en el sistema | |

Tabla 3. Historia de usuario **Encontrar Modelo**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CASO DE PRUEBA** | | | | | |
| **Historia de Usuario** | HU1 | | | | |
| **Desarrollador** | Carlos Daniel Vilaseca | | | | |
| **Probador** |  | | | | |
| **Fecha** |  | | | | |
| **Combinaciones de valores de entrada** | | | **Técnica de diseño empleada** | **Resultados Esperados** | **Resultados reales** |
| **CP** | **Nombres de Variables de entrada (V1, V2, …, Vn)** | **Para cada variable de entrada:**  **variable=valor** |
| 1 | (id, models) | id= ”company-001”  models= [] | Caja Negra | Se lanza la excepción IdNotFoundException y se muestra la ventana de error. |  |
| 2 |  | id= ”company-001”  models= [“company-001”,  ”jobpost-001”] |  | Se retorna una el modelo con el id “company-001” |  |
| 3 |  | id= ”company-002”  models=  [“company-001”, “company-002”,  ”jobpost-001”] |  | Se retorna el modelo con el id “company-002” |  |
| 4 |  | id= ”company-003”  models=  [“company-001”, “company-002”,  ”jobpost-001”] |  | Se lanza la excepción IdNotFoundException y se muestra la ventana de error |  |

Tabla 4. Casos de Prueba de caja negra del método getModelWithID() de la clase Agency.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identificador** | **Nombre HU** | **Prioridad** | | **Puntos estimados** |
| HU2 | Validación de carné Candidato |  | |  |
| **Usuario:** | | | | |
| **Descripción:** Se necesita validar el carné de identidad (ID) de un candidato | | | | |
| **Programador Responsable:** Carlos Daniel Vilaseca | | | | |
| **Escenarios** | | | | |
| **Nombre** | **Condición** | | | **Resultado esperado** |
| El ID no contiene solo números | Pattern.*matches*("\\d+", id); | | | Se lanza la excepcion InvalidIDException y se muestra la ventana de error |
| el ID no tiene 11 digitos | id.length() == 11 | | | Se lanza la excepcion InvalidIDException y se muestra la ventana de error |
| Los dígitos correspondientes a la fecha de nacimiento del candidato no son válidos | El tercer y cuarto dígitos que conforman el mes de nacimiento deben ser mayor que 01 y menor que 12, en cuanto al quinto y sexto dígitos que conforman el día del mes de nacimiento, si el mes es 01,03,05,07,08,10 o 12, entonces el día se debe encontrar entre 01 y 31, si el mes es 04,06,09 u 11, entonces el día se debe encontrar entre 01 y 30, en el caso de que sea 02 el mes se validó que el día se encuentre entre 01 y 29. | | | Se Lanza la excepción InvalidIDException y se muestra el mensaje de error. |
| El ID entrado es valido | Si el id no esta registrado, son solo números, son 11 caracteres y los dígitos correspondientes a la fecha de nacimiento son correctos | | | Se registra el ID correctamente. |
| **Variables** | | | | |
| **Nombre** | **Tipo o Dominio** | | **Descripción** | |
| id | String | | Representa el identificador del candidato | |

Tabla 5. Historia de usuario **Validar carné de candidato**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CASO DE PRUEBA** | | | | | |
| **Historia de Usuario** | HU2 | | | | |
| **Desarrollador** | Carlos Daniel Vilaseca | | | | |
| **Probador** |  | | | | |
| **Fecha** |  | | | | |
| **Combinaciones de valores de entrada** | | | **Técnica de diseño empleada** | **Resultados Esperados** | **Resultados reales** |
| **CP** | **Nombres de Variables de entrada (V1, V2, …, Vn)** | **Para cada variable de entrada:**  **variable=valor** |
| 1 | (id) | id= ”a0106056848” | Caja Negra | Se lanza la excepción InvalidIDEcxception y se muestra la ventana de error. |  |
| 2 |  | id= ”010605684811” |  | Se lanza la excepción InvalidIDEcxception y se muestra la ventana de error. |  |
| 3 |  | id= ”01999968481” |  | Se lanza la excepción InvalidIDEcxception y se muestra la ventana de error. |  |
| 4 |  | id= ”01060568481” |  | Se guarda el id correctamente. |  |

Tabla 5. Casos de Prueba de caja negra del método setID() de la clase Candidate.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identificador** | **Nombre HU** | **Prioridad** | | **Puntos estimados** |
| HU3 | Validación de Teléfono |  | |  |
| **Usuario:** | | | | |
| **Descripción:** Se necesita validar el Teléfono de una Compañía o Candidato | | | | |
| **Programador Responsable:** Carlos Daniel Vilaseca | | | | |
| **Escenarios** | | | | |
| **Nombre** | **Condición** | | | **Resultado esperado** |
| El teléfono no contiene solo números | Pattern.matches("\\d+", phone) | | | Se lanza la excepción InvalidPhoneException y se muestra la ventana de error |
| El teléfono no tiene exactamente 8 dígitos | phone.length() == 8 | | | Se lanza la excepción InvalidPhoneException y se muestra la ventana de error |
| El teléfono es valido | tiene 8 caracteres y son todos números | | | se guarda el teléfono correctamente |
| **Variables** | | | | |
| **Nombre** | **Tipo o Dominio** | | **Descripción** | |
| phone | String | | Representa el teléfono de una Compañía o Candidato | |

Tabla 6. Historia de usuario **Validar teléfono de candidato y compañía.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CASO DE PRUEBA** | | | | | |
| **Historia de Usuario** | HU3 | | | | |
| **Desarrollador** | Carlos Daniel Vilaseca | | | | |
| **Probador** |  | | | | |
| **Fecha** |  | | | | |
| **Combinaciones de valores de entrada** | | | **Técnica de diseño empleada** | **Resultados Esperados** | **Resultados reales** |
| **CP** | **Nombres de Variables de entrada (V1, V2, …, Vn)** | **Para cada variable de entrada:**  **variable=valor** |
| 1 | (phone) | “a7271005” | Caja Negra | Se lanza la excepción InvalidPhoneException y se muestra la ventana de error. |  |
| 2 |  | “727100530” |  | Se lanza la excepción InvalidPhoneException y se muestra la ventana de error. |  |
| 3 |  | 72710053 |  | Se guarda el teléfono correctamente |  |

Tabla 7. Casos de Prueba de caja negra del método setPhone() de las clases Candidate y Company.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identificador** | **Nombre HU** | **Prioridad** | | **Puntos estimados** |
| HU4 | Insertar modelo al sistema |  | |  |
| **Usuario:** | | | | |
| **Descripción:** Se necesita dado un modelo, de cualquier tipo, insertarlo en la lista correspondiente | | | | |
| **Programador Responsable:** Carlos Daniel Vilaseca | | | | |
| **Escenarios** | | | | |
| **Nombre** | **Condición** | | | **Resultado esperado** |
| el modelo es instancia de Candidate | m instance of Candidate | | | Se añade el modelo m a la lista de candidatos |
| el modelo es instancia de Company | m instance of Company | | | Se añade el modelo m a la lista de compañías |
| el modelo es instancia de JobPost | m instance of JobPost | | | Se añade el modelo m a la lista de ofertas de trabajo |
| el modelo es instancia de Interview | m instance of Interview | | | Se añade el modelo m a la lista de entrevistas |
| el modelo no es instancia de Candidate, Company, JobPost o Interview | !(m instance of Candidate || m instance of Company || m instance of JobPost || m instance of Interview) | | | No se guarda en ninguna lista |
| **Variables** | | | | |
| **Nombre** | **Tipo o Dominio** | | **Descripción** | |
| m | Model🡪 clase abstracta | | clase extendida por las clases Candidate, Company, JobPost, Interview | |

Tabla 8. Historia de usuario **Insertar el modelo en la lista correcta.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CASO DE PRUEBA** | | | | | |
| **Historia de Usuario** | HU4 | | | | |
| **Desarrollador** | Vicente Samuel Garófalo | | | | |
| **Probador** |  | | | | |
| **Fecha** |  | | | | |
| **Combinaciones de valores de entrada** | | | **Técnica de diseño empleada** | **Resultados Esperados** | **Resultados reales** |
| **CP** | **Nombres de Variables de entrada (V1, V2, …, Vn)** | **Para cada variable de entrada:**  **variable=valor** |
| 1 | m | m instance of Candidate | Caja Negra | Se añade el modelo m a la lista de candidatos |  |
| 2 |  | m instance of Company |  | Se añade el modelo m a la lista de compañías. |  |
| 3 |  | m instance of JobPost |  | Se añade el modelo m a la lista de ofertas de trabajo |  |
| 4 |  | m instance of Interview |  | Se añade el modelo m a la lista de entrevistas |  |
| 5 |  | m instance of Carro |  | No se añade a ninguna lista |  |

Tabla 9. Casos de Prueba de caja negra del método insertModel() de la clase Agency

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identificador** | **Nombre HU** | **Prioridad** | | **Puntos estimados** |
| HU5 | Establecer un salario a una oferta de trabajo |  | |  |
| **Usuario:** | | | | |
| **Descripción:** Se necesita establecer un salario a las ofertas de trabajo que entren al sistema | | | | |
| **Programador Responsable:** Vicente Samuel Garófalo | | | | |
| **Escenarios** | | | | |
| **Nombre** | **Condición** | | | **Resultado esperado** |
| el salario es mayor que 0 | salary > 0 | | | se anade el salario a la oferta de trabajo correctamente |
| el salario es menor o igual a 0 | salary <= 0 | | | Se lanza la excepcion InvalidSalaryException |
| **Variables** | | | | |
| **Nombre** | **Tipo o Dominio** | | **Descripción** | |
| salary | Numero flotante positivo | | represente el salario correspondiente a una oferta de trabajo que entra al sistema | |

Tabla 10. Historia de usuario **Insertar el modelo en la lista correcta.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CASO DE PRUEBA** | | | | | |
| **Historia de Usuario** | HU5 | | | | |
| **Desarrollador** | Vicente Samuel Garófalo | | | | |
| **Probador** |  | | | | |
| **Fecha** |  | | | | |
| **Combinaciones de valores de entrada** | | | **Técnica de diseño empleada** | **Resultados Esperados** | **Resultados reales** |
| **CP** | **Nombres de Variables de entrada (V1, V2, …, Vn)** | **Para cada variable de entrada:**  **variable=valor** |
| 1 | salary | 0 | Caja Negra | Se lanza la excepción InvalidSalaryException y se muestra el mensaje de error |  |
| 2 |  | 100 |  | Se guarda correctamente el salario correspondiente a la oferta de trabajo |  |

Tabla 11. Casos de Prueba de caja negra del método setSalary() de la clase JobPost.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identificador** | **Nombre HU** | **Prioridad** | | **Puntos estimados** |
| HU6 | Establecer una fecha para la entrevista |  | |  |
| **Usuario:** | | | | |
| **Descripción:** Se necesita establecer una fecha para las entrevistas, y esta no debe ser antes de la fecha actual | | | | |
| **Programador Responsable:** Vicente Samuel Garófalo | | | | |
| **Escenarios** | | | | |
| **Nombre** | **Condición** | | | **Resultado esperado** |
| la fecha es una fecha posterior a la actual. | Calendar today calendar = Calendar.*getInstance*(); Date today date = new Date(today\_calendar.get(Calendar.*YEAR*),(today\_calendar.get(Calendar.*MONTH*)+1),today\_calendar.get(Calendar.*DAY\_OF\_MONTH*));  **date.after(today\_date)** | | | Se establece la fecha de la entrevista correctamente |
| la fecha es anterior o igual a la fecha actual | Calendar today calendar = Calendar.*getInstance*(); Date today date = new Date(today\_calendar.get(Calendar.*YEAR*),(today\_calendar.get(Calendar.*MONTH*)+1),today\_calendar.get(Calendar.*DAY\_OF\_MONTH*));  **!date.after(today\_date)** | | | Se lanza la excepcion InvalidDateException y se muestra el mensaje de error |
| **Variables** | | | | |
| **Nombre** | **Tipo o Dominio** | | **Descripción** | |
| today\_date | tipo Date | | Representa la fecha actual | |
| date | tipo Date | | representa la fecha de la entrevista en cuestión | |

Tabla 12. Historia de usuario **Establecer una fecha correcta a las entrevistas.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CASO DE PRUEBA** | | | | | |
| **Historia de Usuario** | HU6 | | | | |
| **Desarrollador** | Vicente Samuel Garófalo | | | | |
| **Probador** |  | | | | |
| **Fecha** |  | | | | |
| **Combinaciones de valores de entrada** | | | **Técnica de diseño empleada** | **Resultados Esperados** | **Resultados reales** |
| **CP** | **Nombres de Variables de entrada (V1, V2, …, Vn)** | **Para cada variable de entrada:**  **variable=valor** |
| 1 | date, today\_date | 30/6/2022,  26/6/2022 | Caja Negra | Se establece la fecha 30/6/2022 como fecha de la entrevista |  |
| 2 |  | 26/6/2022,  26/6/2022 |  | Se lanza la excepcion InvalidDateException y se muestra el mensaje de error |  |
| 3 |  | 25/6/2022  26/6/2022 |  | Se lanza la excepcion InvalidDateException y se muestra el mensaje de error |  |

Tabla 13. Casos de Prueba de caja negra del método setDate() de la clase Interview.

##### Pruebas unitarias con Junit

A screenshot of a computer

Description automatically generated with low confidence

Img. 10 Prueba Unitaria al método modelExists() de la clase Agency con JUnit.

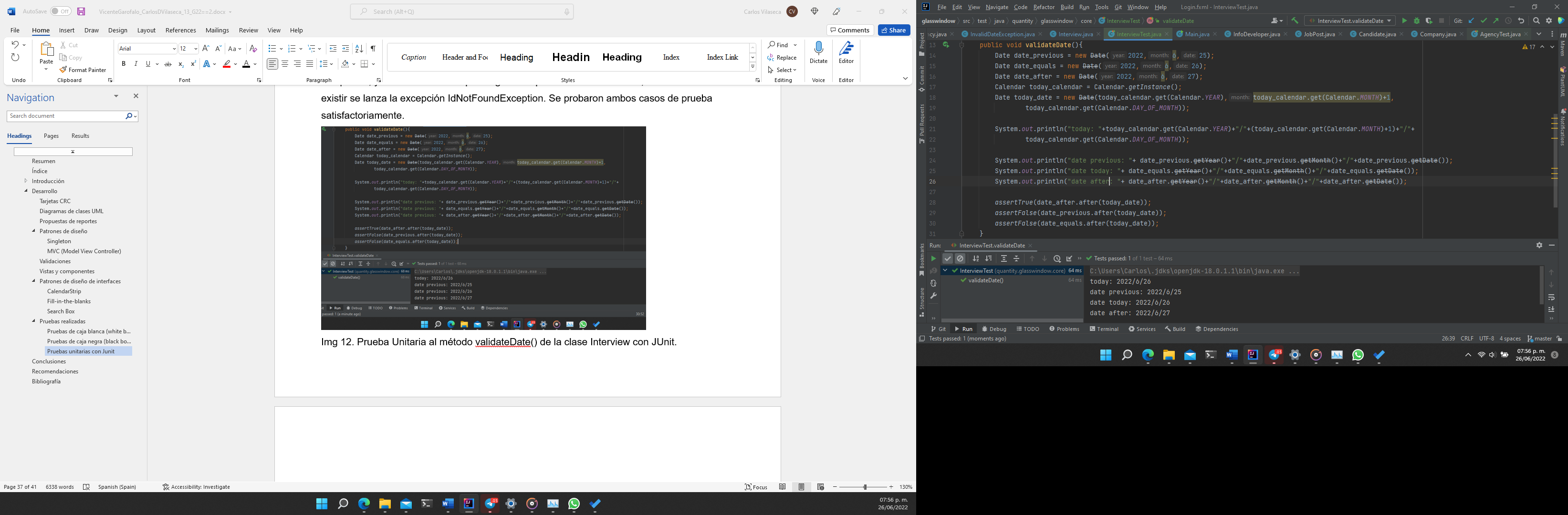
Se realizaron pruebas unitarias al método *modelExists()* de la clase *Agency* (véase img. 10), lo que permitió confirmar el correcto funcionamiento del método. El método en cuestión recibe como parámetro un *String* que hace referencia al ID del modelo que se quiere comprobar, y retorna *true* o *false,* en dependencia de si existe un modelo *(Company, Candidate, JobPost, Interview)* con ese ID o no. Se probaron los dos casos de prueba para el caso en que retorna *true* y para el caso *false.*

A screenshot of a computer

Description automatically generated with low confidence

Img. 11 Prueba Unitaria al método getModelWIthID() de la clase Agency con JUnit.

Se realizaron pruebas unitarias al método *getModelWithID()* de la clase *Agency* (véase img. 11), lo que permitió confirmar el correcto funcionamiento del método. El método en cuestión recibe como parámetro un *String* que hace referencia al ID del modelo que se quiere comprobar, y retorna el modelo que tenga el ID que recibió si este existe, en caso de no existir se lanza la excepción IdNotFoundException. Se probaron ambos casos de prueba satisfactoriamente.



Img 12. Prueba Unitaria al método validateDate() de la clase Interviewcon JUnit.

Se realizaron pruebas unitarias al método *validateDate()* de la clase *Interview* (véase img. 12), lo que permitió confirmar el correcto funcionamiento del método. El método en cuestión recibe como parámetro un objeto de tipo *Date*, que hace referencia a la fecha que se quiere establecer para la entrevista, y retorna *true* o *false* en dependencia de si la fecha que se quiere establecer es posterior a la fecha actual (retorna *true)* o si es igual o anterior a la fecha actual (retorna false). Se probaron los tres casos de prueba satisfactoriamente.

### **Conclusiones**

### **Recomendaciones**

//pressman (tests, components, views), thinking in java, refactor guru+book,

//validación de carnet

<http://www.cubadebate.cu/especiales/2018/12/20/respuesta-a-matematizando-con-el-carne-de-identidad/>

//patrones de diseno de interfaces

[TD\_08110\_15.pdf (uci.cu)](https://repositorio.uci.cu/bitstream/123456789/7311/1/TD_08110_15.pdf) Módulo para la gestión de patrones de diseño de Recursos Educativos Abiertos Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas, referencia de la UCI

### **Bibliografía**