

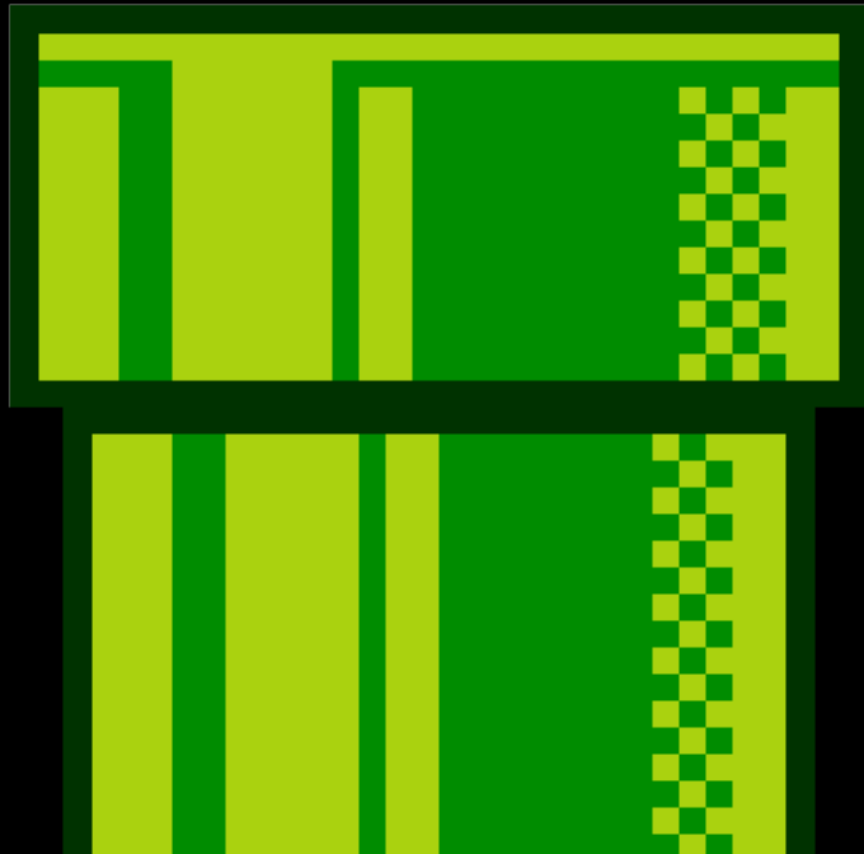
Tarea 0

Modelado y Programación



Integrantes:

Carlos Cruz Rangel
Victor Hugo Gallegos Mota



NOTA: Todas las instrucciones para ejecutar de nuestro programa están declarados en el README.md

Tarea 0

Existen muchas estructuras de datos que podrían implementarse sin embargo nosotros optamos por trabajar esta practica con la estructura de datos Array (T[]). Que aunque el número de elementos es fijo se requerirán subíndices, sabemos que no resulta ser la mejor estructura de datos en términos de Menor/Mejor crecimiento.

En programación de computadoras, Array es una de las estructuras de datos más simples y más utilizadas. En cualquier lenguaje de programación, las matrices tienen algunos puntos en común:

- El contenido de la matriz se almacena usando memoria contigua.
- Todos los elementos de la matriz deben ser del mismo tipo o un tipo derivado del tipo. Por lo tanto, las matrices se consideran estructuras de datos Homogéneas.
- Se puede acceder directamente a los elementos de la matriz. Por ejemplo, si necesita acceder al elemento i -ésimo de la matriz, puede usar directamente `arrayName[i]` para acceder a él.

La complejidad en tiempo usual al trabajar con esta estructura de datos permanece $O(n)$ "Lineal" en la mayoría de casos y continuación analizaremos los métodos asignados en nuestra App.

Añadir elemento al final de la lista:

```
1  /**
2   * Agrega un objeto Empleado al final de la lista
3   *
4   * @param objeto Una instancia de la clase Empleado
5   */
6  public void nuevoReg(Empleado objeto) {
7      Empleado[] arregloTemp = new Empleado[regemp.length + 1];
8      for (int i = 0; i < regemp.length; i++)
9          arregloTemp[i] = regemp[i];
10     arregloTemp[regemp.length] = objeto;
11     regemp = arregloTemp;
12 }
```

Imagen descriptiva de nuestra código de la app sobre añadir elementos al final de la lista

Accede a un elemento de la matriz con complejidad $O(n)$, por lo que el tiempo de acceso a la matriz es constante, en otras palabras, no existe una relación directa con el número de elementos contenidos en la matriz y el tiempo para acceder a un elemento es el mismo.

Debido a que se necesita un solo paso para acceder a un elemento de una matriz a través de su índice, o agregar un elemento al final de una matriz, la complejidad para acceder a un valor en una matriz es $O(1)$.

No hay forma confiable de agregar un elemento al final del arreglo sin recorrer el mismo. Por lo tanto se tendrá que cruzar a través del arreglo, esto es la parte fundamental que nos dará la complejidad $O(n)$ ya que es un array ordenado.

Buscar un elemento en la lista

```
1  /**
2   * Buscar un objeto Empleado en la lista
3   *
4   * @param str Cadena a buscar pos Determina la posición desde donde se buscará
5   *           en el arreglo
6   * @return i Devuelve la posición del elemento buscado
7   */
8  public int buscar(String str, int pos) {
9      String nombre;
10
11     // si str es nula, regresar -1
12     if (str == null)
13         return -1;
14
15     if (pos < 0)
16         pos = 0;
17
18     /**
19      * Averiguar si str esta contenida en el campo nombre de la clase Empleado ¿str
20      * está contenida en nombre? indexOf(String str) Devuelve la posición del
21      * elemento buscado De no encontrar el elemento regresar -1
22      */
23     for (int i = pos; i < regemp.length; i++) {
24         nombre = regemp[i].getNombre();
25         if (nombre.indexOf(str) > -1)
26             return i;
27     }
28     return -1;
29 }
```

Imagen descriptiva del código del programa donde se declara el buscador de elementos en la lista.

En este algoritmo de búsqueda deseamos determinar la posición de nuestro elemento desde donde se buscara, devolviéndonos así la posición del elemento.

Como en el ejemplo anterior debido a que se necesita un solo paso para acceder a un elemento de una matriz a través de su índice, o agregar/eliminar un elemento al final de una matriz, la complejidad para acceder a un valor en una matriz es $O(1)$. Considerando que, buscar linealmente a través de una matriz, a través de su índice, como se vio antes, tiene una complejidad de $O(n)$.

Eliminar elemento de una lista

```
1  /**
2   * Elimina un objeto Empleado de la lista utilizando el número de empleado como
3   * criterio de selección.
4   *
5   * @param numemp Número de empleado del registro del trabajador a eliminar Se
6   *               elimina el objeto Empleado
7   */
8  public boolean eliminarReg(int numemp) {
9      /**
10     * Busca en la lista el objeto Empleado que tiene el número de empleado a
11     * eliminar. Remueve el elemento de la lista Devuelve true si encontró y eliminó
12     * el elemento Devuelve false si no eliminó el elemento
13     */
14     for (int i = 0; i < regemp.length; i++)
15         if (regemp[i].getNumEmp() == numemp) {
16             regemp = borrarReg(i);
17             return true;
18         }
19     return false;
20 }
```

Imagen descriptiva del código para eliminar un elemento de la lista

Se busca eliminar un elemento de la lista utilizando su número de empleado como criterio de selección, si lo encuentra y elimina devuelve true en caso contrario obtenemos false, ejemplo particular parecido a agregar solo que con mas condiciones sin embargo su complejidad continuaría siendo la misma por hacerlo en orden lineal $O(n)$

Crear un nuevo arreglo y eliminar su registro

```
1  /**
2   * Crea un nuevo arreglo de Empleado saltando el elemento n.
3   *
4   * @param n Indice del elemento del arreglo a remover
5   */
6  public Empleado[] borrarReg(int n) {
7      // escribir instrucciones
8      Empleado[] arregloTemp = new Empleado[regemp.length - 1];
9      for (int i = 0; i < regemp.length; i++) {
10         if (i < n)
11             arregloTemp[i] = regemp[i];
12         if (i > n)
13             arregloTemp[i - 1] = regemp[i];
14     }
15     return arregloTemp;
16 }
```

Imagen descriptiva del código donde se crea un nuevo arreglo se elimina el registro

Se busca Crea un nuevo arreglo temporal de Empleado saltando el elemento n. Utilizando el índice del elemento del arreglo a remover.

Eliminar de una posición particular, tendremos que mover todos los $n-1$ elementos restantes, dándonos 0 (n) para nuestro peor caso.

Busca un registro utilizando el número de empleado como condición

```
1
2 /**
3  * Busca un registro utilizando el número de empleado como condición, obteniendo
4  * la posición del elemento encontrado o devolviendo -1 en caso de que el
5  * registro no exista.
6  *
7  * @param bus Guarda el valor recibido como condición (número de empleado) de
8  *           búsqueda
9  * @return
10 */
11 public int buscarParaCam(int bus) {
12     int numeroE;
13     if (bus <= 0)
14         return -1;
15     for (int i = 0; i < regemp.length; i++) {
16         numeroE = regemp[i].getNumEmp();
17         if (numeroE == bus)
18             return i;
19     }
20     return -1;
21 }
```

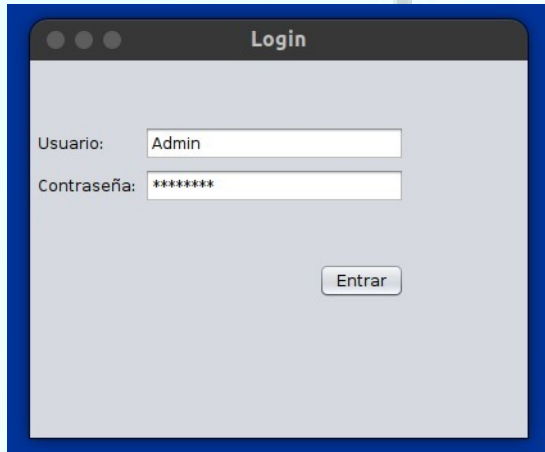
Imagen descriptiva del código del programa donde se busca un registro usando una condición.

Se busca el registro utilizando el número de empleado como condición y así obtener la posición del elemento encontrado o devolviendo -1 en caso de que el registro no exista guardaremos el valor recibido como condición (número de empleado) de búsqueda.

Al igual que el método declarado mas arriba considerando que, buscar linealmente a través de una matriz, a través de su índice, como se vio antes, tiene una complejidad de $O(n)$.

Acompañamiento visual de la funcionalidad del programa.

NOTA: Esto no es un instructivo formal de la ejecución del programa, solo es una vista rápida de la ejecución de programa como introducción a su funcionalidad. EL instructivo y mas información se encuentra en el archivo README.md

A screenshot of a Java Swing window titled "Login". It contains two text input fields: "Usuario:" with the text "Admin" and "Contraseña:" with masked characters "*****". Below the fields is a button labeled "Entrar".

Usuario: Admin

Contraseña: *****

Entrar

Ventana del login, aqui ingresaras tu usuario y contraseña.

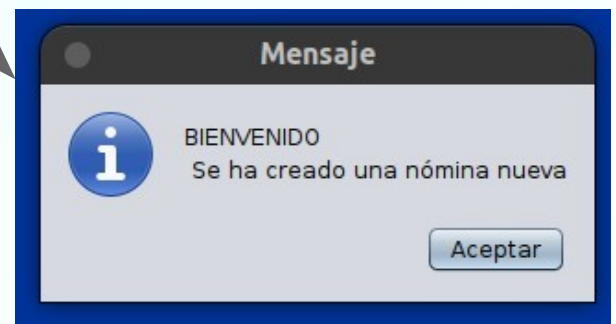
A screenshot of a Java Swing window titled "Mensaje". It features an information icon (a lowercase 'i' in a blue circle) on the left. To the right of the icon, the text reads "Contraseña INCORRECTA" followed by "vuelve a intentarlo" on the next line. At the bottom right is a button labeled "Aceptar".

Mensaje

Contraseña INCORRECTA
vuelve a intentarlo

Aceptar

Si la contraseña es incorrecta se lanzara este aviso y tendrás otro intento para ingresarla.

A screenshot of a Java Swing window titled "Mensaje". It features an information icon (a lowercase 'i' in a blue circle) on the left. To the right of the icon, the text reads "BIENVENIDO" followed by "Se ha creado una nómina nueva" on the next line. At the bottom right is a button labeled "Aceptar".

Mensaje

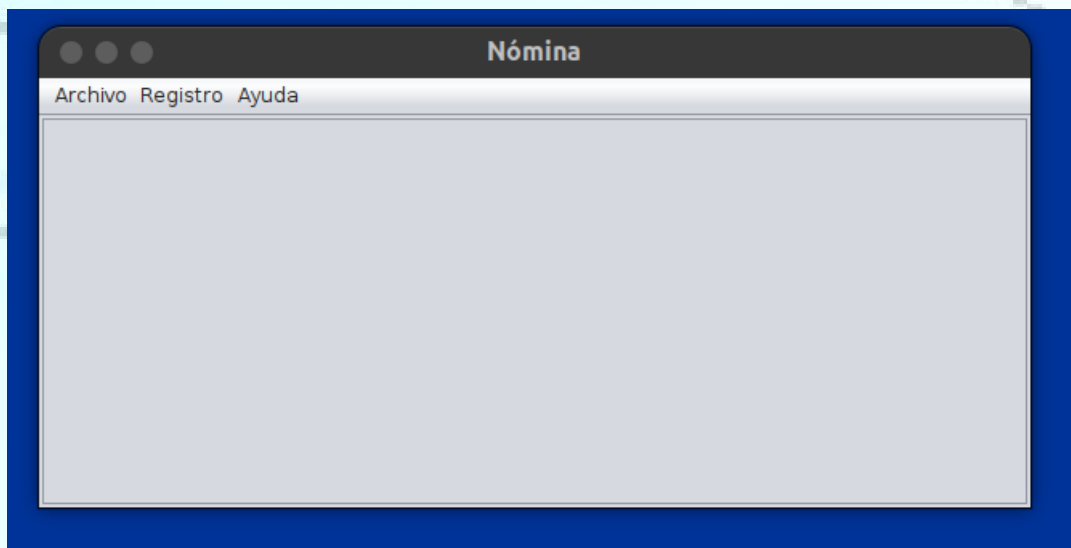
BIENVENIDO
Se ha creado una nómina nueva

Aceptar

Es el mensaje de bienvenida al usuario, este se lanza cuando tu contraseña es correcta.

```
+ Nomina git:(rama_de_victor) x cd /home/hp/Documentos/GitHub/Modelado-y-programacion/Nomina ; /usr/bin/env /usr/lib/jvm/default-java/bin/java -Dfile.encoding=UTF-8 -cp /home/hp/.config/Code/User/workspaceStorage/9b2a7a52e224663ff4fffd39a7088829/redhat.java/jdt_ws/Nomina_e06a7541/bin FormLogin
Contraseña encriptada: 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
```

Una vez introducida la contraseña y dando ENTER se mostrara en consola la encriptación de la contraseña en SHA 256.



Una vez que tengamos la password correcta accedemos al menú principal con 3 opciones: Archivo, Registro y Ayuda.

A screenshot of a macOS-style application window titled "Altas". The window has a dark title bar with a red window control button on the left. The main content area is a light gray form with the following fields and controls:

- Nombre(s):
- Apellido paterno:
- Apellido materno:
- Número de trabajador:
- CURP:
- Dirección:
- Sueldo diario: \$
- Días laborados:
- Correo Electronico:
- Años de antigüedad:
- Puesto:

At the bottom right of the form are two buttons: a light blue "Guardar" button and a red "Cancelar" button.

Si accedemos a la opción Registro y Dar de alta nos abrirá este menú donde se hará el registro de los empleados, empezando por preguntar la misma información general a todos y al final una opción de elegir un puesto para generar mas preguntas dependiendo de su tipo de puesto

Altas

Nombre(s):

Victor Hugo

Apellido paterno:

Gallegos

Apellido materno:

Mota

Número de trabajador:

316160456

CURP:

AMV991024HDFLTC01

Dirección:

Av del rosál s/n Edif Mani N2 depto 201 CDMX

Sueldo diario: \$

200

Días laborados:

15

Correo Electrónico:

316160456@ciencias.unam.mx

Años de antigüedad:

2

Puesto:

Ayudante

Facultad donde imparte clase:

Facultad de Ciencias

Porcentaje de créditos:

100

Nivel de ayudante:

A

Pasante o Titulado:

Titulado

Clase que imparte

Ayudante de Laboratorio Modelado y programación

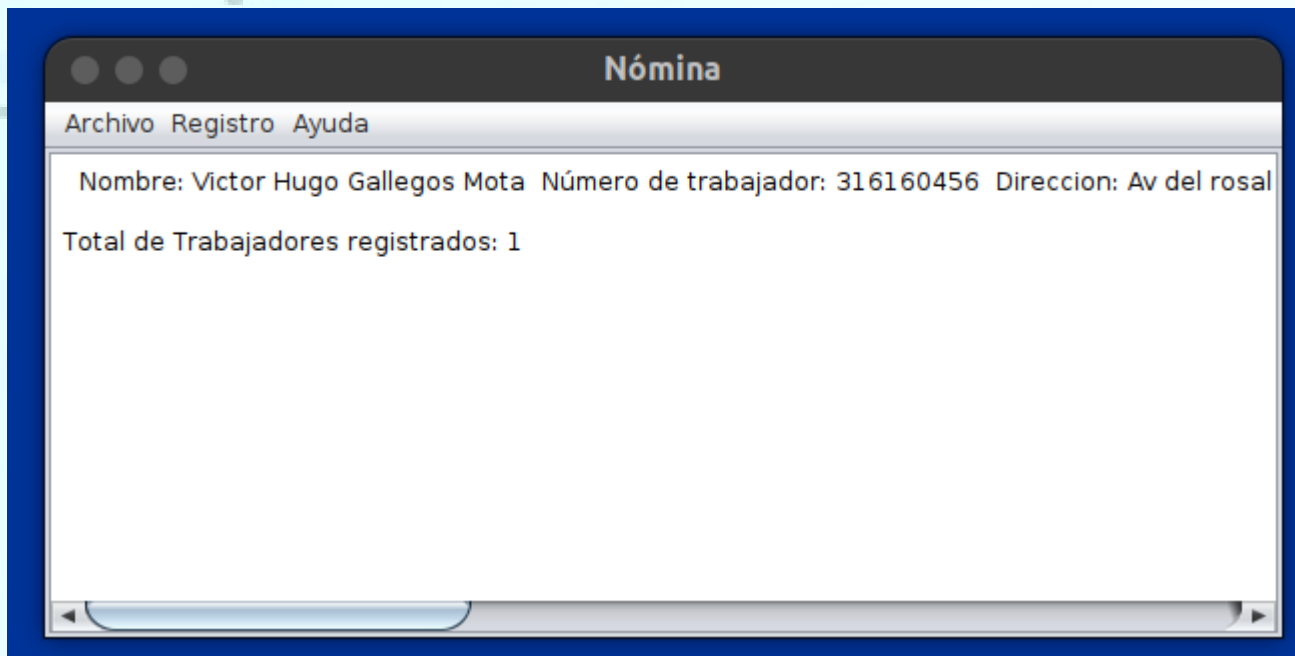
Horario de la clase:

12 a 2 pm

Guardar

Cancelar

Ejemplo de llenado de Altas, una vez llenado los formularios si se desea guardar presionaremos el botón azul de lo contrario el rojo



Si se desea ver a los trabajadores registrados debemos ir a la sección Registro/Mostrar todos y obtendremos esta vista

Se recomienda visitar las demás ventanas para descubrir su funcionamiento, la App es un programa Robusto

Por ultimo Para guardar y generar todos nuestros cambios en un .txt debemos ir a la sección de Archivo, Guardar o Salir esto guardara todo y generara un archivo nomina.txt que servirá cuando se vuelva a abrir la aplicación para cargar la ultima nomina.



Archivo nomina.txt generado después de guardar y salir del programa