INSTITUTO NACIONAL DE APRENDIZAJE (INA)

CENTRO DE FORMACIÓN TURRIALBA

PROGRAMADOR (A) DE APLICACIONES INFORMÁTICAS

MÓDULO 6: PROGRAMACIÓN DE APLICACIONES EMP. EN AMBIENTE WEB

PROYECTO # 1

ESTUDIANTE: ANGY MELANIA CALDERÓN HIDALGO

CÉDULA: 305230269

ROYDEN JAFET CECILIANO HIDALGO

CÉDULA: 702880233

PROFESOR: SERGIO PÉREZ MÉNDEZ

11 FEBRERO DEL 2022

Presentación

Como parte del primer proyecto del módulo Programación de Aplicaciones Empresariales en Ambiente Web, del programa Programador (a) de Aplicaciones Informáticas, se presenta una aplicación para un instituto de enseñanza hipotético: Albert Einstein. La aplicación tiene como objetivo principal, registrar los docentes, cursos y estudiantes del centro, así como demás gestiones relacionadas.

El proyecto se desarrolla mediante el lenguaje JAVA y utilizando el Entorno de Desarrollo Integrado NetBeans (la aplicación se ejecuta por consola). En cuanto al código, este se divide en dos grandes partes: administrador y usuario. A grandes rasgos, el proyecto se desarrolló por medio de Programación Orientada a Objetos, métodos JAVA y manipulación de listas.

Para acceder a administrador se hace una autenticación por contraseña y luego se muestran cada una de las acciones que se pueden realizar en esta sección, las cuales son: administración de cursos, estudiantes y docentes, listar los cursos que imparte un docente a elegir y listar los cursos que lleve un estudiante, de igual manera, seleccionado por el usuario.

Asimismo, las opciones de administración de cursos, estudiantes y docentes muestran una serie de gestiones como agregar, modificar y eliminar los mismos.

Por otra parte, en la sección de usuario, se pueden mostrar los cursos, docentes y estudiantes registrados, así como matricular cursos.

Objetivo General

Registrar datos de estudiantes, docentes y cursos del centro educativo Albert Einstein mediante una aplicación por consola programada en JAVA.

Objetivos Específicos

1. Realizar un registro de la información óptimo, mediante el uso objetos Estudiante, Docente y Curso.
2. Programar un CRUD (crear, leer, actualizar y eliminar, por sus siglas en inglés) para objetos Estudiante, Docente y Curso.
3. Utilizar listas para almacenar objetos Estudiante, Docente y Curso.
4. Programar una sección de matrícula de cursos por estudiante, según se desee.

Problema Empresarial

La escuela comercial ALBERT EINSTEIN tiene una oferta formativa bastante amplia. Debido a esto, cuentan con gran cantidad de docentes.

Hasta ahora, toda la información se ha manipulado en forma física, inclusive los distintos procesos: matrícula, constancias de estudios, reclutamiento de personal, etc. Por tanto, es necesario que usted diseñe un sistema que permita llevar un control de los docentes, estudiantes, cursos, matrícula y el personal encargado de realizar las matrículas y demás actividades administrativas.

La aplicación debe ofrecer 2 cuentas, como usuario y como administrador.

La cuenta de usuario sólo debe permitir consultar la lista de cursos, profesores y estudiantes, así como realizar la matrícula.

La cuenta de administrador debe permitir crear, editar, eliminar la lista de cursos, profesores, estudiantes, así como mostrar cuales cursos está impartiendo un docente específico y cuales cursos matriculó un estudiante en particular.

Aspectos a considerar:

- un docente sólo puede impartir un curso que esté en la oferta
- el curso sólo puede ser impartido por un docente que esté en planilla
- un estudiante sólo puede matricular cursos que estén en la oferta

La aplicación debe permitir que el usuario puede realizar varios procesos según lo desee.

Determina las necesidades

El proyecto requiere el manejo de clases mediante el paradigma de Programación Orientada a Objetos; por lo cual, se crea la clase *Persona* la cual sirve como clase padre de las clases *Estudiante y Docente*. Además, para el control de los cursos se crea la clase Curso.

Una vez creadas estas clases, se inicia la interacción con el usuario a través de la función principal. Al iniciar la aplicación se solicita el tipo de usuario a utilizar, el cual se obtiene del método *menuGeneral()* el cual maneja el control excepciones ante posibles respuestas no esperadas por el sistema, después de guardar la opción seleccionada el sistema entra en una estructura *switch* donde se puede ingresar a las opciones del usuario normal o a las de administrador.

En caso de entrar al usuario estándar, se llama a la función *usuarioMenu()* el cual muestras las siguientes opciones: *lista de cursos, lista de profesores, lista de estudiantes y matrícula.* Este método retorna la opción elegida en un dato de tipo byte que se utiliza al llamar al método *gestionesUsuario(byte opc),* el cual contiene una estructura *switch* que se encarga de redirigir el programa al método seleccionado según la opción guardada en la propiedad opc; los posibles metodos son los siguientes: *imprimirCursos(), imprimirDocentes(), imprimirEstudiantes() y matricula();* cada uno con un nombre apropiado para su respectiva función.

Se considera importante mencionar que el método matricula lista los estudiantes registrados y solicita la selección de uno, para luego agregar dentro de este objeto los cursos que desea matricular a través de una lista de tipo curso (*List<Curso> cursos*).

Por otro lado; se tiene el perfil de administrador, el cual valida la contraseña con el método booleano *validacionAdmin()* en el cual se da al usuario tres intentos para ingresar de manera correcta su contraseña, en caso de gastar los tres intentos la función devuelve *false,* caso contrario retorna *true.* Al devolver false, la aplicación regresa a *menuGeneral().*

Si la función *validacionAdmin()* retorna *true,* la aplicación entra al método *administradorMenu()* donde se muestran las siguientes opciones: *Administración cursos, Administración profesores, Administración estudiantes, Cursos por docente y Cursos por estudiante.* Si el usuario elije alguna de las primeras tres opciones, podrá agregar, modificar y eliminar objetos de su respectiva clase; si elija alguna de los otras dos, podrá visualizar los cursos por docente o estudiante, respectivamente.

Además de los métodos mencionados; se crean otros para la validación de formatos y entradas de usuario, los cuales son vistos a profundidad en la sección Documentación de la aplicación programada.

Limitaciones

La realización de este proyecto cubre la mayoría de los temas tratados durante el presente módulo, por lo cual se considera que cumple a cabalidad con los requerimientos establecidos en enunciado del proyecto; sin embargo, presenta ciertas limitaciones, las cuales se enumeran a continuación:

1. La información registrada se mantiene mientras el programa esté en ejecución, pero en cuanto este acabe, los datos se perderán, dado que no se almacenan en un archivo o una base de datos. Por lo tanto, la utilización de este programa no sería práctico para el registro real de datos en un negocio de educación.
2. Aunque se hacen validaciones según los posibles errores de entrada de datos y manejo de los mismos mediante *try catch,* no se cuentan con pruebas específicas de testeo, que permitan prever una gran cantidad de errores relacionados.
3. La manipulación del programa por consola no es estándar para este tipo de aplicaciones. Se espera que cuente con una interfaz gráfica que mejore la usabilidad y genera una experiencia de usuario óptima.

Código de la solución

```
package system.registration.ina;

import Models.Curso;

import Models.Docente;
```

```java
import Models.Estudiante;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;
import java.util.InputMismatchException;
import java.util.List;
import java.util.Locale;
import java.util.Scanner;
import static jdk.nashorn.internal.objects.NativeString.toUpperCase;

public class SystemRegistrationIna {

static String password = "cheeseburger09"; // Contraseña para acceso a Administrador.
static Scanner scan = new Scanner(System.in); // Instancia de la clase Scanner para ingresar datos.

// Declaración de listas a usar.
static List<Docente> docentes = new ArrayList<>();
static List<Curso> cursos = new ArrayList<>();
static List<Estudiante> estudiantes = new ArrayList<>();

/**
* @param args the command line arguments
*/
public static void main(String[] args) {
byte tipoUsuario, opc;
```

```
do {
tipoUsuario = menuGeneral();


switch (tipoUsuario) {
case 1:
// Se dirige al usuario al menú de acciones para Usuario.
do {
opc = usuarioMenu();
gestionesUsuario(opc);
} while (opc != 5);
break;
case 2:
// Se valida el ingreso a Administrador.
boolean entra = validacionAdmin();
if (entra) {
// Se dirige al usuario al menú de acciones para Administrador.
do {
opc = administradorMenu();
gestionesAdministrador(opc);
} while (opc != 6);
}
break;
case 3:
System.out.print("\n\n\tFin de la ejecución del programa.");
break;
default:
System.out.print("\n\n\t[ Error ] - Elija una opción válida");
break;
}
```

```
} while (tipoUsuario == 1 || tipoUsuario == 2);
}

// Menú general del programa.
static byte menuGeneral() {
byte opcion = 0;
boolean error = false;

do {
error = false; // Se inicializa al entrar al ciclo, sino entra en loop infinito
System.out.print("\n\t[ Bienvenido (a) al sistema de matrícula ]"
+ "\n\t[1] - Usuario"
+ "\n\t[2] - Administrador"
+ "\n\t[3] - Salir"
+ "\n\n\tElija una opción: ");
try {
opcion = scan.nextByte();
if ((opcion < 1) && (opcion > 3)) {
opcion = 0;
System.out.print("\n\tIngrese una opción válida entre 1 y 3");
error = true;
}
} catch (InputMismatchException ex) {
error = true;
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (error);
```

```
return opcion;
}

// Menú con las gestiones disponibles para el usuario.
static byte usuarioMenu() {
byte opcion = 0;
boolean error = false;

do {
error = false;
System.out.print("\n\t[ GESTIONES DE USUARIO ]"
+ "\n\t[1] - Lista de cursos"
+ "\n\t[2] - Lista de profesores"
+ "\n\t[3] - Lista de estudiantes"
+ "\n\t[4] - Matrícula"
+ "\n\t[5] - Salir"
+ "\n\n\tElija una opción: ");
try {
opcion = scan.nextByte();
if ((opcion < 1) && (opcion > 4)) {
opcion = 0;
System.out.print("\n\tIngrese una opción válida entre 1 y 5");
error = true;
}
} catch (InputMismatchException ex) {
error = true;
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
```

```
}
} while (error);


return opcion;
}


// Se realizan acciones de acuerdo con la selección del usuario en la sección de
// Usuario.
static void gestionesUsuario(byte opc) {
switch (opc) {
case 1:
imprimirCursos();
break;
case 2:
imprimirDocentes();
break;
case 3:
imprimirEstudiantes();
break;
case 4:
matricula();
break;
default:
break;


}
}


// Método que valida la contraseña ingresada para administrador.
```

```
static boolean validacionAdmin() {
byte intentos = 3;

System.out.print("\n\tIngrese la contraseña: ");
String seguir = scan.nextLine();

do {
String contraIngresada = scan.nextLine();

if (contraIngresada.equals(password)) {
return true;
} else {
intentos--;
System.out.print("\n\t CONTRASEÑA INCORRECTA. Intentos restantes: " + intentos + " \n");
}
} while (intentos != 0);

return false;

}

// Menú con las gestiones disponibles para el administrador.
static byte administradorMenu() {

byte opcion = 0;
boolean error = false;
```

```
System.out.print("\n\t[ GESTIONES DE ADMINISTRADOR ]"
+ "\n\t[1] - Administración cursos"
+ "\n\t[2] - Administración profesores"
+ "\n\t[3] - Administración estudiantes"
+ "\n\t[4] - Cursos por docente"
+ "\n\t[5] - Cursos por estudiante"
+ "\n\t[6] - Salir"
+ "\n\n\tPresione ENTER para continuar... ");


do {
String seguir = scan.nextLine();
error = false; // Se inicializa al entrar al ciclo, sino entra en loop infinito
try {
System.out.print("\n\n\tSeleccione: ");
opcion = scan.nextByte();
if ((opcion < 1) || (opcion > 6)) {
opcion = 0;
System.out.print("\n\tIngrese una opción válida entre 1 y 6\n");
error = true;
}
} catch (InputMismatchException ex) {
error = true;
System.out.print("\n\t[ Error ] - Ingrese un número válido\n");
scan.nextLine();
}
} while (error);


return opcion;
```

```
}

// Se realizan acciones de acuerdo con la
// selección del administardor.
static void gestionesAdministrador(byte opc) {

byte opc2 = 0;

switch (opc) {
case 1:
// Se redirige al menú de gestiones referentes a los cursos.
System.out.print("\n\t[ ADMINISTRACIÓN CURSOS ]");
opc2 = administracionClases();
gestionCursos(opc2);
break;
case 2:
// Se redirige al menú de gestiones referentes a los profesores.
System.out.print("\n\t[ ADMINISTRACIÓN PROFESORES ]");
opc2 = administracionClases();
gestionDocentes(opc2);
break;
case 3:
// Se redirige al menú de gestiones referentes a los estudiantes.
System.out.print("\n\t[ ADMINISTRACIÓN ESTUDIANTES ]");
opc2 = administracionClases();
gestionEstudiantes(opc2);
break;
case 4:
// Se muestran cursos de un docente específico
```

```
buscarCursosDocente();
break;
case 5:
// Se muestran cursos de un estudiante específico
buscarCursosEstudiante();
break;
default:
System.out.print("\n\t[ Error ] - Ingrese una opción válida");
break;


}
}


// Método general que muestra las acciones que se pueden realizar con Cursos,
// Docentes y Estudiantes. Retorna la opción seleccionada por el usuario
static byte administracionClases() {
byte opcion = 0;
boolean error = false;


System.out.print("\n\t[1] - Agregar"
+ "\n\t[2] - Eliminar"
+ "\n\t[3] - Editar"
+ "\n\t[4] - Salir");


do {
error = false; // Se inicializa al entrar al ciclo, sino entra en loop infinito
try {
System.out.print("\n\n\tSeleccione: ");
opcion = scan.nextByte();
```

```
if ((opcion < 1) || (opcion > 4)) {
opcion = 0;
System.out.print("\n\tIngrese una opción válida entre 1 y 4");
error = true;
}
} catch (InputMismatchException ex) {
error = true;
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (error);

return opcion;
}

// Métodos para los cursos

/**
* Redirige a la acción seleccionada por el usuario
*
* @param opc acción a realizar por el usuario
*/
static void gestionCursos(byte opc) {
switch (opc) {
case 1:
agregarCurso();
break;
case 2:
eliminarCurso();
```

```
break;
case 3:
modificarCurso();
break;
default:
break;
}
}

// Método que agrega los cursos a la lista correspondiente.
static void agregarCurso() {
Curso curso = new Curso();
short num = 0;
scan.nextLine();
System.out.print("\n\tNombre del curso: ");
curso.setNombre(scan.nextLine());

do {
System.out.print("\tCantidad máxima de estudiantes: ");
try {
num = scan.nextShort();
if (num < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);
```

```
curso.setCupoMax(num);
num = 0;


do {
System.out.print("\tCréditos del curso: ");
try {
num = scan.nextByte();
if (num < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);


curso.setCreditos((byte) num);
num = 0;


if (!docentes.isEmpty()) {
do {
imprimirDocentes();
System.out.print("\n\tNúmero del docente que impartirá el curso: ");
try {
num = scan.nextShort();
if (num < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
```

```
num = 0;
} else if (num > docentes.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);


curso.setDocenteEncargado(docentes.get(num - 1));
} else
System.out.print("\n\t[ NO hay docentes registrados para dirigir el curso ]");


cursos.add(curso);
System.out.print("\n\t[ Cambios realizados con éxito ]\n");
}


// Método que modifica un curso específico.
static void modificarCurso() {
Curso curso;
short num = 0;


if (!cursos.isEmpty()) {
do {
imprimirCursos();
System.out.print("\n\tNúmero del curso a modificar: ");
try {
```

```
num = scan.nextShort();
if (num < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
num = 0;
} else if (num > cursos.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);

curso = cursos.get(num - 1);
cursos.remove(num - 1);

scan.nextLine();
System.out.print("\n\tNombre del curso: ");
curso.setNombre(scan.nextLine());

do {
System.out.print("\tCantidad máxima de estudiantes: ");
try {
num = scan.nextShort();
if (num < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
num = 0;
}
```

```
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);

curso.setCupoMax(num);
num = 0;

do {
System.out.print("\tCréditos del curso: ");
try {
num = scan.nextByte();
if (num < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);

curso.setCreditos((byte) num);
num = 0;

if (!docentes.isEmpty()) {
do {
imprimirDocentes();
```

```
System.out.print("\n\tNúmero del docente que impartirá el curso: ");
try {
num = scan.nextShort();
if (num < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
num = 0;
} else if (num > docentes.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);


curso.setDocenteEncargado(docentes.get(num - 1));
} else
System.out.print("\n\t[ NO hay docentes registrados para dirigir el curso ]");
cursos.add(curso);
System.out.print("\n\t[ Cambios realizados con éxito ]\n");
} else
System.out.print("\n\t[ NO hay cursos registrados ]\n\n");


}


// Método que elimina un curso seleccionado.
static void eliminarCurso() {
short num = 0;
```

```
if (!cursos.isEmpty()) {
do {
imprimirCursos();
System.out.print("\n\tNúmero del curso a eliminar: ");
try {
num = scan.nextShort();
if (num < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
num = 0;
} else if (num > cursos.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);


try {
cursos.remove(num - 1);
System.out.print("\n\t[ Cambios realizados con éxito ]\n");
} catch (Exception ex) {
System.out.print("\n\t[ Error ] - Ha ocurrido un error, curso NO eliminado");
}
} else {
System.out.print("\n\t[ NO hay cursos registrados ]");
}
}
```

```
// Método que ordena los cursos por nombre y de forma ascendente.
static void ordenarCursos() {
Comparator<Curso> orderByName = (p1, p2) -> p1.getNombre().compareTo(p2.getNombre());
Collections.sort(cursos, orderByName);
}

// Método que imprime los cursos por nombre y de forma ascendente.
static void imprimirCursos() {
if (!cursos.isEmpty()) {
ordenarCursos();
short count = 1;
for (Curso curso : cursos) {
System.out.print("\n\t[" + (count++) + "]\n" + curso.toString());
}
System.out.println("");
} else
System.out.print("\n\t[ NO hay cursos registrados ]");
}

// Métodos para los estudiantes

/**
* Redirige a la acción seleccionada por el usuario
*
* @param opc acción a realizar por el usuario
*/
static void gestionEstudiantes(byte opc) {
switch (opc) {
```

```
case 1:
agregarEstudiante();
break;
case 2:
eliminarEstudiante();
break;
case 3:
modificarEstudiante();
break;
default:
break;
}
}

// Método que agrega los estudiantes a la lista correspondiente.
static void agregarEstudiante() {
Estudiante estud = new Estudiante();
short num = 0;
scan.nextLine();
System.out.print("\n\tNombre: ");
estud.setNombre(scan.nextLine());
System.out.print("\tApellidos: ");
estud.setApellidos(scan.nextLine());
System.out.print("\tCédula: ");
estud.setCedula(validarCedula());
System.out.print("\tDirección actual: ");
estud.setDireccion(scan.nextLine());
System.out.print("\tCorreo electrónico: ");
estud.setCorreo(scan.nextLine());
```

```
do {
System.out.print("\tEdad: ");
try {
num = scan.nextByte();
if (num < 15 || num > 100) {
System.out.print("\n\t[ Error ] - Ingrese un número entre 15 - 100");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);

estud.setEdad((byte) num);
num = 0;

do {
System.out.print("\tGrado actual: ");
try {
num = scan.nextByte();
if (num < 1 || num > 10) {
System.out.print("\n\t[ Error ] - Ingrese un número entre 1 - 10");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);
```

```
estud.setGradoActual((byte) num);
num = 0;

scan.nextLine();
System.out.print("\tRequiere adecuación? (S/N): ");
estud.setAdecuacion(validarSN());

System.out.print("\tRequiere solicitar beca? (S/N): ");
estud.setBeca(validarSN());

estudiantes.add(estud);
System.out.print("\n\t[ Cambios realizados con éxito ]\n");
}

// Método que elimina un estudiante específico.
static void eliminarEstudiante() {
short num = 0;

if (!estudiantes.isEmpty()) {
do {
imprimirCursos();
System.out.print("\n\tNúmero del estudiante a eliminar: ");
scan.nextLine();
try {
num = scan.nextShort();
if (num < 1) {
```

```
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
num = 0;
} else if (num > estudiantes.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);


try {
cursos.remove(num - 1);
System.out.print("\n\t[ Cambios realizados con éxito ]\n");
} catch (Exception ex) {
System.out.print("\n\t[ Error ] - Ha ocurrido un error, estudiante NO eliminado");
}
} else
System.out.print("\n\t[ NO hay estudiantes registrados ]\n");


}


// Método que modifica un estudiante seleccionado.
static void modificarEstudiante() {
short num = 0;
short index = 0;


if (!estudiantes.isEmpty()) {
```

```
do {
imprimirEstudiantes();
System.out.print("\n\tNúmero del estudiante a modificar: ");
scan.nextLine();
try {
index = scan.nextShort();
if (index < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
index = 0;
} else if (index > estudiantes.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido");
index = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (index == 0);


index = (short) (index - 1);


scan.nextLine();
System.out.print("\n\tNombre: ");
estudiantes.get(index).setNombre(scan.nextLine());
System.out.print("\tApellidos: : ");
estudiantes.get(index).setApellidos(scan.nextLine());
System.out.print("\tCédula: ");
estudiantes.get(index).setCedula(scan.nextLine());
System.out.print("\tDirección actual: ");
estudiantes.get(index).setDireccion(scan.nextLine());
```

```
System.out.print("\tCorreo electrónico: ");
estudiantes.get(index).setCorreo(scan.nextLine());

do {
System.out.print("\tEdad: ");
try {
num = scan.nextByte();
if (num < 15 || num > 100) {
System.out.print("\n\t[ Error ] - Ingrese un número entre 15 - 100");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);

estudiantes.get(index).setEdad((byte) num);
num = 0;

do {
System.out.print("\tGrado actual: ");
try {
num = scan.nextByte();
if (num < 1 && num > 10) {
System.out.print("\n\t[ Error ] - Ingrese un número entre 1 - 10");
num = 0;
}
} catch (InputMismatchException ex) {
```

```
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);

estudiantes.get(index).setGradoActual((byte) num);
num = 0;

scan.nextLine();
System.out.print("\tRequiere adecuación? (S/N): ");
estudiantes.get(index).setAdecuacion(validarSN());

System.out.print("\tRequiere solicitar beca? (S/N): ");
estudiantes.get(index).setBeca(validarSN());

System.out.print("\n\t[ Estudiante modificado con éxito ]");
} else
System.out.print("\n\t[ NO hay estudiantes registrados ]\n");

}

// Método que ordena los estudiantes por nombre en orden ascendente.
static void ordenarEstudiantes() {
Comparator<Estudiante> orderByName = (p1, p2) -> p1.getApellidos().compareTo(p2.getApellidos());
Collections.sort(estudiantes, orderByName);
}
```

```
// Método que imprime los estudiantes por nombre en orden ascendente.
static void imprimirEstudiantes() {
ordenarEstudiantes();
short count = 1;
if (!estudiantes.isEmpty()) {
for (Estudiante estudiante : estudiantes) {
System.out.print("\n\t[" + (count++) + "]" + estudiante.toString());
}
} else
System.out.print("\n\t[ NO hay estudiantes registrados ]\n");
}

static void gestionDocentes(byte opc2) {
Docente doc = new Docente();
switch (opc2) {
case 1:
doc = agregarDocente();
docentes.add(doc);
break;
case 2:
eliminarDocente();
break;
case 3:
modificarDocente();
break;
default:
break;
}
}
```

```
// Método que agrega docentes a la lista correspondiente
static Docente agregarDocente() {
boolean error = false;
ArrayList<Curso> cursosDocente = new ArrayList<>();
Curso newMateria = new Curso();

String nombre, apellidos, direccion, cedula, correo, gradoAcad;
boolean colegiado = false;
boolean propiedad = false;
byte edad;
Date fechaInicio;

do {
error = false;
nombre = "";
apellidos = "";
cedula = "";
direccion = "";
correo = "";
edad = 0;
gradoAcad = "";
fechaInicio = null;

try {
System.out.print("\n\tNombre:");
scan.nextLine();
nombre = scan.nextLine();
System.out.print("\tApellidos:");
apellidos = scan.nextLine();
```

```
System.out.print("\tCédula:");
cedula = validarCedula(); // Se valida que en cédula se ingresen datos válidos
System.out.print("\tDirección:");
direccion = scan.nextLine();
System.out.print("\tCorreo:");
correo = scan.nextLine();
System.out.print("\tEdad:");
edad = validarEdad(); // Se valida que en edad se ingresen datos válidos
System.out.print("\tSeleccione la materia que imparte: ");
imprimirCursos();
int num = 0;


if (!cursos.isEmpty()) {
do {
try {
num = scan.nextInt();
if (num == 0) {
System.out.print("\n\t[Error 1] - Ingrese datos válidos");
} else {
if ((num - 1) > cursos.size()) {
System.out.print("\n\t[Error 2] - Ingrese datos válidos");
num = 0;
}
}
} catch (InputMismatchException e) {
System.out.print("\n\t[Error 3] - Ingrese datos válidos");
}


} while (num == 0);
```

```
newMateria = cursos.get(num - 1);
cursosDocente.add(newMateria); // Se agrega la materia al arraylist local del objeto.

} else {
System.out.print("\n\t[ NO hay docentes registrados ]\n");
}

scan.nextLine();
System.out.print("\tGrado académico: ");
gradoAcad = scan.nextLine();
System.out.print("\t¿Es colegiado? (s/n)");
colegiado = validarSN(); // Se obtiene verdadero o falso de acuerdo con la respuesta del usuario.
System.out.print("\t¿Está en propiedad? (s/n)");
propiedad = validarSN();
System.out.print("\tFecha de inicio en la institución (DD/MM/YYYY)");
fechaInicio = validarFecha();

} catch (Exception e) {
System.out.print("\n\t[ Error ] - ingrese datos válidos. \n");
error = true;
}

} while (error == true);

Docente newdocente = new Docente(cursosDocente, gradoAcad, colegiado, propiedad, fechaInicio,
nombre, apellidos,
cedula, direccion, correo, edad);
```

```
System.out.print("\n\t[ Cambios realizados con éxito ]\n");


return newdocente;
}


// Método que valida las respuestas en formato S/N
static boolean validarSN() {
boolean error = false;


do {
error = false;
String word = scan.nextLine();
word = toUpperCase(word); // Se pasa la respuesta a mayúsculas.
if (!"S".equals(word) && !"N".equals(word)) {
System.out.print("[Error] - Ingrese datos correctos.");
error = true;
} else {
if ("S".equals(word)) {
return true;
}
}
} while (error);


return false;
}


// Método que valida la cédula.
static String validarCedula() {
```

```
boolean error = false;
String cedula = "";


do {
error = false;
do {
error = false;
try {
cedula = scan.nextLine();
Integer.parseInt(cedula);
int digitos = cedula.length();


if (digitos != 10) {
System.out.print("[ERROR] - Ingrese la cédula en formato de 10 dígitos.\n");
error = true;
}


} catch (Exception e) {
error = true;
System.out.print("[ERROR] - Ingrese solo 10 números.\n");
}
} while (error);
for (Docente doc : docentes) {
if (doc.getCedula().equals(cedula))
error = true;
}
for (Estudiante est : estudiantes) {
if (est.getCedula().equals(cedula))
```

```
error = true;
}
} while (error);
return cedula;
}

// Método que valida la edad.
static byte validarEdad() {

boolean error = false;
byte edad = 0;

do {
error = false;
try {
edad = scan.nextByte();

if (edad < 0 || edad > 99) {
System.out.print("[ERROR] - Ingrese datos válidos.");
error = true;
}

} catch (Exception e) {
error = true;
System.out.print("[ERROR] - Ingrese sólo números.");
scan.nextLine();
}
} while (error);
```

```
return edad;
}

// Método que ordena por apellido los docentes, de forma ascendente.
static void ordenarDocentes() {
Comparator<Docente> orderByName = (p1, p2) -> p1.getApellidos().compareTo(p2.getApellidos());
Collections.sort(docentes, orderByName);
}

// Método que imprime los docentes.
static void imprimirDocentes() {
System.out.print("\n\t[ DOCENTES REGISTRADOS ]");
ordenarDocentes();
Calendar cal = Calendar.getInstance();
String fechaStr;
int l = 0;
if (!docentes.isEmpty()) {
for (Docente doc : docentes) {
fechaStr = "";
cal.setTime(doc.getInicioInstitucion());
fechaStr = String.valueOf(cal.get(Calendar.DAY_OF_MONTH)) + "/"
+ String.valueOf(cal.get(Calendar.MONTH) + "/" + String.valueOf(cal.get(Calendar.YEAR)));
l++;
System.out.print("\n\t[" + (l) + "]\n");
System.out.print("\tNombre: " + doc.getNombre() + "\n");
System.out.print("\tApellidos " + doc.getApellidos() + "\n");
System.out.print("\tCédula " + doc.getCedula() + "\n");
System.out.print("\tDirección " + doc.getDireccion() + "\n");
```

```
System.out.print("\tCorreo: " + doc.getCorreo() + "\n");
System.out.print("\tEdad: " + doc.getEdad() + "\n");
System.out.print("\tColegiado: " + doc.isColegiado() + "\n");
System.out.print("\tPropiedad: " + doc.isPropiedad() + "\n");
System.out.println("\tFecha de inicio en la institución: " + fechaStr + "\n");
// Se imprimen los cursos que imparte un profesor determinado
if (!doc.getMateriasImparte().isEmpty()) {
System.out.print("\tMATERIAS QUE IMPARTE\n");
for (int i = 0; i < doc.getMateriasImparte().size(); i++) {
System.out.print("\t" + doc.getMateriasImparte().get(i).getNombre() + "\n");
}
} else {
System.out.print("\n\tDocente no asignado a ninguna materia\n");
}



}
}
}


// Método que elimina un objeto docente.
static void eliminarDocente() {
byte num = 0;


if (!docentes.isEmpty()) {
do {
imprimirDocentes();
System.out.print("\n\tNúmero del docente a eliminar: ");
try {
num = scan.nextByte();
```

```
if (num == 0) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0\n");
} else if ((num - 1) > docentes.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido\n");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido\n");
scan.nextLine();
}
} while (num == 0);


try {
docentes.remove(num - 1);
System.out.print("\n\tDocente eliminado con éxito\n");
} catch (Exception ex) {
System.out.print("\n\t[ Error ] - Ha ocurrido un error, docente NO eliminado");
}
} else {
System.out.print("\n\t[ NO hay docentes registrados ]\n");
}



}


// Método que modifica un docente.
static void modificarDocente() {
int num = 0;
if (!docentes.isEmpty()) {
do {
```

```
imprimirDocentes();
System.out.print("\n\tNúmero del docente a modificar: ");
try {
num = scan.nextByte();
if (num == 0) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0\n");
} else if ((num - 1) > docentes.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido\n");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido\n");
scan.nextLine();
}
} while (num == 0);

Docente doc = new Docente();
doc = agregarDocente();

try {
docentes.remove(num - 1);
docentes.add(num - 1, doc);
} catch (Exception ex) {
System.out.print("\n\t[ Error ] - Ha ocurrido un error, docente NO modificado\n");
}
} else {
System.out.print("\n\t[ NO hay docentes registrados ]\n");
}
```

```
}

// Método que muestra los cursos por docente específico.
static void buscarCursosDocente() {
int num = 0;
System.out.print("\n\tCURSOS POR DOCENTE");
do {
imprimirDocentes();
System.out.print("\n\tNúmero del docente a consultar: ");
try {
num = scan.nextByte();
if (num == 0) {
System.out.print("\n\t[ Error 1] - Ingrese un número mayor a 0");
} else if (num > docentes.size()) {
System.out.print("\n\t[ Error 2] - Elija un número válido");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error 3] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);

for (int i = 0; i < docentes.get(num - 1).getMateriasImparte().size(); i++) {
System.out.print("\n\t Curso: " + docentes.get(num - 1).getMateriasImparte().get(i).getNombre() + "\n");
}
}

// Método que muestra los cursos por estudiante específico.
```

```
static void buscarCursosEstudiante() {
int num = 0;
System.out.print("CURSOS POR ESTUDIANTE");
do {
imprimirEstudiantes();
System.out.print("\n\tNúmero del estudiante a consultar: ");
try {
num = scan.nextByte();
if (num == 0) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
} else if ((num - 1) > estudiantes.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);

for (int i = 0; i < estudiantes.get(num - 1).getCursos().size(); i++) {
System.out.print("\tCurso: " + estudiantes.get(num - 1).getCursos().get(i).getNombre() + "\n");
}

}

/**
* Método para realizar matrícula de estudiantes
*/
```

```
static void matricula() {
boolean seguir = true;
short num = 0;
int index = 0;
List<Curso> list = new ArrayList<>();

if (!estudiantes.isEmpty() && !cursos.isEmpty()) {
do {
imprimirEstudiantes();
System.out.print("\n\tNúmero del estudiante a matricular: ");
try {
index = scan.nextShort();
if (index == 0) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
} else if (index > estudiantes.size() || index < estudiantes.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido");
index = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (index == 0);
index = (short) (index - 1);

while (seguir) {
imprimirCursos();
do {
System.out.print("\n\tNúmero del curso a añadir: ");
try {
```

```
num = scan.nextShort();
if (num < 1) {
System.out.print("\n\t[ Error ] - Ingrese un número mayor a 0");
num = 0;
} else if (num > cursos.size()) {
System.out.print("\n\t[ Error ] - Elija un número válido");
num = 0;
}
} catch (InputMismatchException ex) {
System.out.print("\n\t[ Error ] - Ingrese un número válido");
scan.nextLine();
}
} while (num == 0);


list.add(cursos.get(num - 1));
System.out.print("\n\t¡Curso matriculado con éxito!");


scan.nextLine();
System.out.print("\n\tDesea matricular otro curso? (S/N): ");
seguir = validarSN();
}
estudiantes.get(index).setCursos(list);
} else if (estudiantes.isEmpty())
System.out.print("\n\t[ NO hay estudiantes registrados ]\n");
else if (cursos.isEmpty())
System.out.print("\n\t[ NO hay cursos registrados ]");
}


// Método que solicita y valida una fecha al usuario.
```

```
static Date validarFecha() {
Date date = null;
String linea;
SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy", new Locale("ES"));


do {
linea = scan.nextLine();
try {
date = formato.parse(linea);
if (!formato.format(date).equals(linea)) {
throw new Exception();
}
} catch (Exception ex) {
System.out.print("\n\t[Error] - Formato no válido, formato requerido: DD/MM/YYYY");
date = null;
}
} while (date == null);


return date;
}
}
```

Documentación de la aplicación programada

En el planeamiento del proyecto, se crean las siguientes clases:

Persona: clase que maneja atributos deseables de las personas. Sirve como clase padre para las clases *Docente y Estudiante.*

Docente: esta clase extiende de la clase *Persona* y representa los profesores (as) que se tratan en el sistema.

Estudiante: esta clase extiende de la clase *Persona* y representa los estudiantes que se tratan en el sistema.

Curso: clase para manejar las materias que imparte el instituto.

Cada uno de estos métodos contienen sus respectivos Setters y Getters, así como el método toString() (algunos).

Clase system-registration-ina

static byte menuGeneral(): Menú general del programa.

static byte usuarioMenu(): Menú con las gestiones disponibles para el usuario.

static void gestionesUsuario(byte opc): Se realizan acciones de acuerdo con la selección del usuario en la sección de Usuario.

static boolean validacionAdmin(): Método que valida la contraseña ingresada para administrador.

static byte administradorMenu(): Menú con las gestiones disponibles para el administrador.

static void gestionesAdministrador(byte opc): Se realizan acciones de acuerdo con la selección del administardor.

static byte administracionClases(): Método general que muestra las acciones que se pueden realizar con Cursos, Docentes y Estudiantes. Retorna la opción seleccionada por el usuario.

static void gestionCursos(byte opc): Redirige a la acción seleccionada por el usuario. Acción a realizar por el usuario.

static void agregarCurso(): Método que agrega los cursos a la lista correspondiente.

static void modificarCurso(): Método que modifica un curso específico.

static void eliminarCurso(): Método que elimina un curso seleccionado.

static void ordenarCursos(): Método que ordena los cursos por nombre y de forma ascendente.

static void imprimirCursos(): Método que imprime los cursos por nombre y de forma ascendente.

static void gestionEstudiantes(byte opc): Redirige a la acción seleccionada por el usuario.

static void agregarEstudiante(): Método que agrega los estudiantes a la lista correspondiente.

static void eliminarEstudiante(): Método que elimina un estudiante específico.

static void modificarEstudiante(): Método que modifica un estudiante seleccionado.

static void ordenarEstudiantes(): Método que ordena los estudiantes por nombre en orden ascendente.

static void imprimirEstudiantes(): Método que imprime los estudiantes por nombre en orden ascendente.

static void gestionDocentes(byte opc2): Redirige a la acción seleccionada por el usuario.

static Docente agregarDocente(): Método que agrega docentes a la lista correspondiente.

static boolean validarSN(): Método que valida las respuestas en formato S/N.

static String validarCedula(): Método que valida la cédula.

static byte validarEdad(): Método que valida la edad.

static void ordenarDocentes(): Método que ordena por apellido los docentes, de forma ascendente.

static void modificarDocente(): Método que modifica un docente.

static void buscarCursosDocente(): Método que muestra los cursos por docente específico.

static void buscarCursosEstudiante(): Método que muestra los cursos por estudiante específico.

static void matricula (): Método para realizar matrícula de estudiantes.

static Date validarFecha(): Método que solicita y valida una fecha al usuario.