import java.util.\*;

import java.text.ParseException;

import java.text.SimpleDateFormat;

public class SistemaGestionProyectos {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

GestorProyectos gestorProyectos = new GestorProyectos();

List<Usuario> usuarios = new ArrayList<>();

usuarios.add(new Usuario("john\_doe", "john123", "Administrador"));

usuarios.add(new Usuario("jane\_smith", "jane456", "Usuario"));

usuarios.add(new Usuario("michael\_brown", "mike789", "Usuario"));

usuarios.add(new Usuario("emily\_jones", "emily101", "Usuario"));

crearProyectosPreregistrados(gestorProyectos);

asignarMiembrosPreregistrados(gestorProyectos);

crearTareasPreregistradas(gestorProyectos);

crearEquiposPreregistrados(gestorProyectos);

while (true) {

System.out.println("\n--- Menú de Inicio ---");

System.out.println("1. Iniciar sesión");

System.out.println("2. Terminar programa");

System.out.print("Seleccione una opción: ");

int opcionInicio = scanner.nextInt();

scanner.nextLine();

if (opcionInicio == 1) {

Usuario usuario = iniciarSesion(scanner, usuarios);

if (usuario == null) {

System.out.println("Inicio de sesión fallido.");

continue;

}

System.out.println("¡Bienvenido, " + usuario.getNombre() + "!");

System.out.println("Nivel de permisos: " + usuario.getNivelPermisos());

menuPrincipal(scanner, gestorProyectos, usuario, usuarios);

} else if (opcionInicio == 2) {

System.out.println("Terminando el programa...");

scanner.close();

return;

} else {

System.out.println("Opción no válida. Intente de nuevo.");

}

}

}

private static void crearProyectosPreregistrados(GestorProyectos gestorProyectos) {

Calendar calendar = Calendar.getInstance();

calendar.add(Calendar.DAY\_OF\_MONTH, -5);

Date fechaInicioAlpha = calendar.getTime();

calendar.add(Calendar.MONTH, 3);

Date fechaFinAlpha = calendar.getTime();

gestorProyectos.crearProyecto("Proyecto Alpha", "Desarrollo de una app móvil", fechaInicioAlpha, fechaFinAlpha, "En progreso", "Alta", true);

calendar.setTime(new Date());

calendar.add(Calendar.DAY\_OF\_MONTH, -10);

Date fechaInicioBeta = calendar.getTime();

calendar.add(Calendar.MONTH, 2);

Date fechaFinBeta = calendar.getTime();

gestorProyectos.crearProyecto("Proyecto Beta", "Desarrollo de una plataforma web", fechaInicioBeta, fechaFinBeta, "En planificación", "Media", true);

}

private static void asignarMiembrosPreregistrados(GestorProyectos gestorProyectos) {

Miembro miembro1 = new Miembro("jane\_smith", "Desarrollador");

Miembro miembro2 = new Miembro("michael\_brown", "Diseñador");

Miembro miembro3 = new Miembro("emily\_jones", "Tester");

gestorProyectos.asignarMiembrosAProyecto("Proyecto Alpha", Arrays.asList(miembro1, miembro2, miembro3), "john\_doe");

}

private static void crearTareasPreregistradas(GestorProyectos gestorProyectos) {

Calendar calendar = Calendar.getInstance();

calendar.add(Calendar.DAY\_OF\_MONTH, -2);

Date fechaLimiteTarea1 = calendar.getTime();

calendar.add(Calendar.DAY\_OF\_MONTH, 5);

Date fechaLimiteTarea2 = calendar.getTime();

calendar.add(Calendar.DAY\_OF\_MONTH, 10);

Date fechaLimiteTarea3 = calendar.getTime();

gestorProyectos.crearTarea("Tarea 1", "Completar el diseño de la interfaz", new Date(), fechaLimiteTarea1, "Alta", "Pendiente");

gestorProyectos.crearTarea("Tarea 2", "Desarrollar la funcionalidad principal", new Date(), fechaLimiteTarea2, "Media", "Pendiente");

gestorProyectos.crearTarea("Tarea 3", "Realizar pruebas de usuario", new Date(), fechaLimiteTarea3, "Baja", "Pendiente");

}

private static void crearEquiposPreregistrados(GestorProyectos gestorProyectos) {

Equipo equipo1 = new Equipo("Equipo de Desarrollo");

equipo1.agregarMiembro(new Miembro("jane\_smith", "Desarrollador"));

equipo1.agregarMiembro(new Miembro("michael\_brown", "Diseñador"));

gestorProyectos.agregarEquipo(equipo1);

Equipo equipo2 = new Equipo("Equipo de Pruebas");

equipo2.agregarMiembro(new Miembro("emily\_jones", "Tester"));

gestorProyectos.agregarEquipo(equipo2);

}

private static void menuPrincipal(Scanner scanner, GestorProyectos gestorProyectos, Usuario usuario, List<Usuario> usuarios) {

while (true) {

System.out.println("\n--- Sistema de Gestión de Proyectos ---");

if (usuario.getNivelPermisos().equals("Administrador")) {

System.out.println("1. Crear equipo");

System.out.println("2. Eliminar equipo");

System.out.println("3. Ver equipos creados");

System.out.println("4. Ver equipos asignados");

System.out.println("5. Crear tarea");

System.out.println("6. Asignar tarea a miembro");

System.out.println("7. Actualizar estado de tarea");

System.out.println("8. Agregar comentario a tarea");

System.out.println("9. Eliminar tarea");

System.out.println("10. Ver tareas");

System.out.println("11. Ver tareas asignadas");

System.out.println("12. Generar informe de progreso");

System.out.println("13. Generar reporte de desempeño");

System.out.println("14. Crear proyecto");

System.out.println("15. Editar proyecto");

System.out.println("16. Eliminar proyecto");

System.out.println("17. Asignar miembros a un proyecto");

System.out.println("18. Asignar tarea a proyecto");

System.out.println("19. Ver características de un proyecto");

System.out.println("20. Ver proyectos asignados");

System.out.println("21. Mensajería");

System.out.println("22. Cambiar nivel de permisos de un miembro");

System.out.println("23. Ajustes de perfil");

System.out.println("24. Crear nuevo miembro");

System.out.println("25. Eliminar miembro");

System.out.println("26. Ver miembros disponibles");

System.out.println("27. Salir");

} else {

System.out.println("1. Crear tarea");

System.out.println("2. Asignar tarea a miembro");

System.out.println("3. Actualizar estado de tarea");

System.out.println("4. Agregar comentario a tarea");

System.out.println("5. Ver características de un proyecto");

System.out.println("6. Mensajería");

System.out.println("7. Ver tareas");

System.out.println("8. Generar informe de progreso");

System.out.println("9. Ajustes de perfil");

System.out.println("10. Ver tareas asignadas");

System.out.println("11. Ver equipos asignados");

System.out.println("12. Ver proyectos asignados");

System.out.println("13. Salir");

}

System.out.print("Seleccione una opción: ");

int opcion = scanner.nextInt();

scanner.nextLine();

switch (opcion) {

case 1:

if (usuario.getNivelPermisos().equals("Administrador")) {

gestorProyectos.crearEquipo(scanner, usuarios);

} else {

gestorProyectos.crearTarea(scanner);

}

break;

case 2:

if (usuario.getNivelPermisos().equals("Administrador")) {

eliminarEquipo(scanner, gestorProyectos);

} else {

gestorProyectos.asignarTareaAMiembro(scanner, usuarios, usuario);

}

break;

case 3:

if (usuario.getNivelPermisos().equals("Administrador")) {

gestorProyectos.verEquipos(scanner);

} else {

gestorProyectos.actualizarEstadoTarea(scanner);

}

break;

case 4:

if (usuario.getNivelPermisos().equals("Administrador")) {

verEquiposAsignados(usuario, gestorProyectos);

} else {

gestorProyectos.agregarComentarioATarea(scanner);

}

break;

case 5:

if (usuario.getNivelPermisos().equals("Administrador")) {

gestorProyectos.crearTarea(scanner);

} else {

gestorProyectos.verCaracteristicasProyecto(scanner, usuarios);

}

break;

case 6:

if (usuario.getNivelPermisos().equals("Administrador")) {

gestorProyectos.asignarTareaAMiembro(scanner, usuarios, usuario);

} else {

menuMensajeria(scanner, gestorProyectos, usuario, usuarios);

}

break;

case 7:

if (usuario.getNivelPermisos().equals("Administrador")) {

gestorProyectos.actualizarEstadoTarea(scanner);

} else {

gestorProyectos.verTareas(scanner);

}

break;

case 8:

if (usuario.getNivelPermisos().equals("Administrador")) {

gestorProyectos.agregarComentarioATarea(scanner);

} else {

gestorProyectos.generarInformeProgreso(scanner);

}

break;

case 9:

if (usuario.getNivelPermisos().equals("Administrador")) {

eliminarTarea(scanner, gestorProyectos);

} else {

ajustesPerfil(scanner, usuario, gestorProyectos);

}

break;

case 10:

if (usuario.getNivelPermisos().equals("Usuario")) {

verTareasAsignadas(usuario, gestorProyectos);

} else {

gestorProyectos.verTareas(scanner);

}

break;

case 11:

if (usuario.getNivelPermisos().equals("Administrador")) {

verTareasAsignadas(usuario, gestorProyectos);

} else {

verEquiposAsignados(usuario, gestorProyectos);

}

break;

case 12:

if (usuario.getNivelPermisos().equals("Administrador")) {

gestorProyectos.generarInformeProgreso(scanner);

} else {

verProyectosAsignados(usuario, gestorProyectos);

}

break;

case 13:

if (usuario.getNivelPermisos().equals("Administrador")) {

gestorProyectos.generarReporteDesempeño(scanner);

} else {

System.out.println("Saliendo al menú de inicio de sesión...");

return;

}

break;

case 14:

gestorProyectos.crearProyecto(scanner);

break;

case 15:

gestorProyectos.editarProyecto(scanner);

break;

case 16:

gestorProyectos.eliminarProyecto(scanner);

break;

case 17:

System.out.println("\n--- Proyectos Disponibles ---");

for (int i = 0; i < gestorProyectos.getProyectos().size(); i++) {

System.out.println((i + 1) + ". " + gestorProyectos.getProyectos().get(i).getNombre());

}

System.out.print("Seleccione el número del proyecto para asignar miembros: ");

int opcionProyecto = scanner.nextInt();

scanner.nextLine();

if (opcionProyecto < 1 || opcionProyecto > gestorProyectos.getProyectos().size()) {

System.out.println("Opción no válida.");

continue;

}

Proyecto proyectoSeleccionado = gestorProyectos.getProyectos().get(opcionProyecto - 1);

gestorProyectos.asignarMiembrosAProyecto(scanner, usuarios, usuario.getNombre(), proyectoSeleccionado);

break;

case 18:

asignarTareaAProyecto(scanner, gestorProyectos);

break;

case 19:

gestorProyectos.verCaracteristicasProyecto(scanner, usuarios);

break;

case 20:

verProyectosAsignados(usuario, gestorProyectos);

break;

case 21:

menuMensajeria(scanner, gestorProyectos, usuario, usuarios);

break;

case 22:

cambiarNivelPermisos(scanner, usuarios);

break;

case 23:

ajustesPerfil(scanner, usuario, gestorProyectos);

break;

case 24:

if (usuario.getNivelPermisos().equals("Administrador")) {

crearNuevoMiembro(scanner, usuarios);

}

break;

case 25:

if (usuario.getNivelPermisos().equals("Administrador")) {

eliminarMiembro(scanner, usuarios, gestorProyectos, usuario);

}

break;

case 26:

verMiembrosDisponibles(usuarios, gestorProyectos);

break;

case 27:

System.out.println("Saliendo al menú de inicio de sesión...");

return;

default:

System.out.println("Opción no válida. Intente de nuevo.");

}

}

}

private static void verTareasAsignadas(Usuario usuario, GestorProyectos gestorProyectos) {

List<Tarea> tareasAsignadas = usuario.getTareasAsignadas(gestorProyectos);

if (tareasAsignadas.isEmpty()) {

System.out.println("No tiene tareas asignadas.");

} else {

System.out.println("--- Tareas Asignadas ---");

for (Tarea tarea : tareasAsignadas) {

System.out.println("- " + tarea.getNombre());

}

}

}

private static void verEquiposAsignados(Usuario usuario, GestorProyectos gestorProyectos) {

List<Equipo> equiposAsignados = usuario.getEquiposAsignados(gestorProyectos);

if (equiposAsignados.isEmpty()) {

System.out.println("No tiene equipos asignados.");

} else {

System.out.println("--- Equipos Asignados ---");

for (Equipo equipo : equiposAsignados) {

System.out.println("- " + equipo.getNombre());

}

}

}

private static void verProyectosAsignados(Usuario usuario, GestorProyectos gestorProyectos) {

List<Proyecto> proyectosAsignados = usuario.getProyectosAsignados(gestorProyectos);

if (proyectosAsignados.isEmpty()) {

System.out.println("No tiene proyectos asignados.");

} else {

System.out.println("--- Proyectos Asignados ---");

for (Proyecto proyecto : proyectosAsignados) {

System.out.println("- " + proyecto.getNombre());

}

}

}

private static void ajustesPerfil(Scanner scanner, Usuario usuario, GestorProyectos gestorProyectos) {

while (true) {

System.out.println("\n--- Ajustes de Perfil ---");

System.out.println("1. Cambiar nombre de usuario");

System.out.println("2. Cambiar contraseña");

System.out.println("3. Volver al menú principal");

System.out.print("Seleccione una opción: ");

int opcion = scanner.nextInt();

scanner.nextLine();

switch (opcion) {

case 1:

String nuevoNombre = scanner.nextLine();

String antiguoNombre = usuario.getNombre();

usuario.setNombre(nuevoNombre);

System.out.println("Nombre de usuario actualizado a: " + nuevoNombre);

actualizarNombreMiembroEnProyectos(antiguoNombre, nuevoNombre, gestorProyectos);

break;

case 2:

System.out.print("Nueva contraseña: ");

String nuevaContrasena = scanner.nextLine();

usuario.setContrasena(nuevaContrasena);

System.out.println("Contraseña actualizada.");

break;

case 3:

return;

default:

System.out.println("Opción no válida. Intente de nuevo.");

}

}

}

private static void actualizarNombreMiembroEnProyectos(String antiguoNombre, String nuevoNombre, GestorProyectos gestorProyectos) {

for (Proyecto proyecto : gestorProyectos.getProyectos()) {

for (Miembro miembro : proyecto.getMiembros()) {

if (miembro.getNombre().equals(antiguoNombre)) {

miembro.setNombre(nuevoNombre);

}

}

for (Tarea tarea : proyecto.getTareas()) {

for (Miembro miembro : tarea.getMiembrosAsignados()) {

if (miembro.getNombre().equals(antiguoNombre)) {

miembro.setNombre(nuevoNombre);

}

}

}

}

for (Equipo equipo : gestorProyectos.getEquipos()) {

for (Miembro miembro : equipo.getMiembros()) {

if (miembro.getNombre().equals(antiguoNombre)) {

miembro.setNombre(nuevoNombre);

}

}

}

}

private static void crearNuevoMiembro(Scanner scanner, List<Usuario> usuarios) {

System.out.print("Nombre de usuario: ");

String nombreUsuario = scanner.nextLine();

System.out.print("Contraseña: ");

String contrasena = scanner.nextLine();

System.out.print("Nivel de permisos (Administrador/Usuario): ");

String nivelPermisos = scanner.nextLine();

if (!nivelPermisos.equalsIgnoreCase("Administrador") && !nivelPermisos.equalsIgnoreCase("Usuario")) {

System.out.println("Nivel de permisos no válido. Debe ser 'Administrador' o 'Usuario'.");

return;

}

usuarios.add(new Usuario(nombreUsuario, contrasena, nivelPermisos));

System.out.println("Nuevo miembro creado: " + nombreUsuario);

}

private static Usuario iniciarSesion(Scanner scanner, List<Usuario> usuarios) {

System.out.print("Nombre de usuario: ");

String nombreUsuario = scanner.nextLine();

System.out.print("Contraseña: ");

String contrasena = scanner.nextLine();

for (Usuario user : usuarios) {

if (user.getNombre().equals(nombreUsuario) && user.getContrasena().equals(contrasena)) {

return user;

}

}

return null;

}

private static void menuMensajeria(Scanner scanner, GestorProyectos gestorProyectos, Usuario usuario, List<Usuario> usuarios) {

while (true) {

System.out.println("\n--- Mensajería ---");

System.out.println("1. Enviar mensaje");

System.out.println("2. Ver mensajes recibidos");

System.out.println("3. Volver al menú principal");

System.out.print("Seleccione una opción: ");

int opcionMensajeria = scanner.nextInt();

scanner.nextLine();

switch (opcionMensajeria) {

case 1:

gestorProyectos.enviarMensaje(scanner, usuarios, usuario);

break;

case 2:

gestorProyectos.verMensajesRecibidos(usuario);

break;

case 3:

return;

default:

System.out.println("Opción no válida. Intente de nuevo.");

}

}

}

private static void asignarTareaAProyecto(Scanner scanner, GestorProyectos gestorProyectos) {

System.out.println("\n--- Tareas Disponibles ---");

for (Tarea tarea : gestorProyectos.getTareas()) {

System.out.println("- " + tarea.getNombre());

}

System.out.print("Seleccione el nombre de la tarea para asignar a un proyecto: ");

String nombreTarea = scanner.nextLine();

Tarea tareaSeleccionada = null;

for (Tarea tarea : gestorProyectos.getTareas()) {

if (tarea.getNombre().equals(nombreTarea)) {

tareaSeleccionada = tarea;

break;

}

}

if (tareaSeleccionada == null) {

System.out.println("Tarea no encontrada.");

return;

}

System.out.println("\n--- Proyectos Disponibles ---");

for (Proyecto proyecto : gestorProyectos.getProyectos()) {

System.out.println("- " + proyecto.getNombre());

}

System.out.print("Seleccione el nombre del proyecto para asignar la tarea: ");

String nombreProyecto = scanner.nextLine();

Proyecto proyectoSeleccionado = null;

for (Proyecto proyecto : gestorProyectos.getProyectos()) {

if (proyecto.getNombre().equals(nombreProyecto)) {

proyectoSeleccionado = proyecto;

break;

}

}

if (proyectoSeleccionado == null) {

System.out.println("Proyecto no encontrado.");

return;

}

proyectoSeleccionado.asignarTareas(Collections.singletonList(tareaSeleccionada));

System.out.println("Tarea " + tareaSeleccionada.getNombre() + " asignada al proyecto " + proyectoSeleccionado.getNombre());

}

private static void cambiarNivelPermisos(Scanner scanner, List<Usuario> usuarios) {

System.out.println("\n--- Miembros Disponibles ---");

for (Usuario usuario : usuarios) {

System.out.println("- " + usuario.getNombre());

}

System.out.print("Seleccione el nombre del miembro para cambiar su nivel de permisos: ");

String nombreMiembro = scanner.nextLine();

Usuario miembroSeleccionado = null;

for (Usuario usuario : usuarios) {

if (usuario.getNombre().equals(nombreMiembro)) {

miembroSeleccionado = usuario;

break;

}

}

if (miembroSeleccionado == null) {

System.out.println("Miembro no encontrado.");

return;

}

System.out.print("Nuevo nivel de permisos (Administrador/Usuario): ");

String nuevoNivelPermisos = scanner.nextLine();

miembroSeleccionado.setNivelPermisos(nuevoNivelPermisos);

System.out.println("Nivel de permisos actualizado para " + nombreMiembro);

}

private static void eliminarTarea(Scanner scanner, GestorProyectos gestorProyectos) {

System.out.println("\n--- Tareas Disponibles ---");

for (int i = 0; i < gestorProyectos.getTareas().size(); i++) {

System.out.println((i + 1) + ". " + gestorProyectos.getTareas().get(i).getNombre());

}

System.out.print("Seleccione el número de la tarea a eliminar: ");

int opcion = scanner.nextInt();

scanner.nextLine();

if (opcion < 1 || opcion > gestorProyectos.getTareas().size()) {

System.out.println("Opción no válida.");

return;

}

Tarea tareaSeleccionada = gestorProyectos.getTareas().get(opcion - 1);

gestorProyectos.eliminarTarea(tareaSeleccionada.getNombre());

System.out.println("Tarea eliminada: " + tareaSeleccionada.getNombre());

}

private static void eliminarEquipo(Scanner scanner, GestorProyectos gestorProyectos) {

System.out.println("\n--- Equipos Disponibles ---");

for (int i = 0; i < gestorProyectos.getEquipos().size(); i++) {

System.out.println((i + 1) + ". " + gestorProyectos.getEquipos().get(i).getNombre());

}

System.out.print("Seleccione el número del equipo a eliminar: ");

int opcion = scanner.nextInt();

scanner.nextLine();

if (opcion < 1 || opcion > gestorProyectos.getEquipos().size()) {

System.out.println("Opción no válida.");

return;

}

Equipo equipoSeleccionado = gestorProyectos.getEquipos().get(opcion - 1);

gestorProyectos.eliminarEquipo(equipoSeleccionado.getNombre());

System.out.println("Equipo eliminado: " + equipoSeleccionado.getNombre());

}

private static void eliminarMiembro(Scanner scanner, List<Usuario> usuarios, GestorProyectos gestorProyectos, Usuario usuario) {

System.out.println("\n--- Miembros Disponibles ---");

for (Usuario u : usuarios) {

if (!u.getNombre().equals(usuario.getNombre())) {

System.out.println("- " + u.getNombre());

}

}

System.out.print("Ingrese el nombre del miembro a eliminar: ");

String nombreMiembroEliminar = scanner.nextLine();

Usuario usuarioSeleccionado = null;

for (Usuario u : usuarios) {

if (u.getNombre().equals(nombreMiembroEliminar)) {

usuarioSeleccionado = u;

break;

}

}

if (usuarioSeleccionado != null) {

usuarios.remove(usuarioSeleccionado);

System.out.println("Miembro eliminado: " + usuarioSeleccionado.getNombre());

gestorProyectos.eliminarMiembroDeProyectos(usuarioSeleccionado.getNombre());

} else {

System.out.println("Miembro no encontrado.");

}

}

private static void verMiembrosDisponibles(List<Usuario> usuarios, GestorProyectos gestorProyectos) {

System.out.println("\n--- Miembros Disponibles ---");

for (Usuario usuario : usuarios) {

System.out.println("- " + usuario.getNombre());

}

System.out.print("Seleccione el nombre del miembro para ver detalles: ");

String nombreMiembro = new Scanner(System.in).nextLine();

Usuario miembroSeleccionado = null;

for (Usuario usuario : usuarios) {

if (usuario.getNombre().equals(nombreMiembro)) {

miembroSeleccionado = usuario;

break;

}

}

if (miembroSeleccionado == null) {

System.out.println("Miembro no encontrado.");

return;

}

System.out.println("Detalles del Miembro: " + miembroSeleccionado.getNombre());

System.out.println("Fecha de Creación: " + new Date());

boolean asignado = false;

for (Proyecto proyecto : gestorProyectos.getProyectos()) {

for (Miembro miembro : proyecto.getMiembros()) {

if (miembro.getNombre().equals(miembroSeleccionado.getNombre())) {

System.out.println("Asignado a Proyecto: " + proyecto.getNombre());

asignado = true;

}

}

for (Tarea tarea : proyecto.getTareas()) {

for (Miembro miembro : tarea.getMiembrosAsignados()) {

if (miembro.getNombre().equals(miembroSeleccionado.getNombre())) {

System.out.println("Asignado a Tarea: " + tarea.getNombre() + " en Proyecto: " + proyecto.getNombre());

asignado = true;

}

}

}

}

for (Equipo equipo : gestorProyectos.getEquipos()) {

for (Miembro miembro : equipo.getMiembros()) {

if (miembro.getNombre().equals(miembroSeleccionado.getNombre())) {

System.out.println("Asignado a Equipo: " + equipo.getNombre());

asignado = true;

}

}

}

if (!asignado) {

System.out.println("No hay asignaciones para este miembro.");

}

}

}

class GestorProyectos {

private List<Proyecto> proyectos = new ArrayList<>();

private List<Miembro> miembros = new ArrayList<>();

private List<Equipo> equipos = new ArrayList<>();

private List<Mensaje> mensajes = new ArrayList<>();

private List<Tarea> tareas = new ArrayList<>();

public void crearProyecto(String nombre, String descripcion, Date fechaInicio, Date fechaFin, String estado, String prioridad, boolean preregistrado) {

Proyecto proyecto = new Proyecto(nombre, descripcion, fechaInicio, fechaFin, estado, prioridad, preregistrado);

proyectos.add(proyecto);

}

public void crearProyecto(Scanner scanner) {

System.out.print("Nombre del proyecto: ");

String nombre = scanner.nextLine();

System.out.print("Descripción: ");

String descripcion = scanner.nextLine();

System.out.print("Fecha de inicio (yyyy-MM-dd): ");

String fechaInicioStr = scanner.nextLine();

Date fechaInicio = parseFecha(fechaInicioStr);

System.out.print("Fecha de finalización (yyyy-MM-dd): ");

String fechaFinStr = scanner.nextLine();

Date fechaFin = parseFecha(fechaFinStr);

System.out.print("Estado: ");

String estado = scanner.nextLine();

System.out.print("Prioridad: ");

String prioridad = scanner.nextLine();

Proyecto proyecto = new Proyecto(nombre, descripcion, fechaInicio, fechaFin, estado, prioridad, false);

proyectos.add(proyecto);

System.out.println("Proyecto creado: " + nombre);

}

private Date parseFecha(String fechaStr) {

SimpleDateFormat formato = new SimpleDateFormat("yyyy-MM-dd");

try {

return formato.parse(fechaStr);

} catch (ParseException e) {

System.out.println("Formato de fecha inválido. Se usará la fecha actual.");

return new Date();

}

}

public void agregarEquipo(Equipo equipo) {

equipos.add(equipo);

}

public void verCaracteristicasProyecto(Scanner scanner, List<Usuario> usuarios) {

System.out.println("\n--- Proyectos Disponibles ---");

for (Proyecto proyecto : proyectos) {

System.out.println("- " + proyecto.getNombre());

}

System.out.print("Seleccione el nombre del proyecto para ver sus características: ");

String nombreProyecto = scanner.nextLine();

Proyecto proyectoSeleccionado = null;

for (Proyecto proyecto : proyectos) {

if (proyecto.getNombre().equals(nombreProyecto)) {

proyectoSeleccionado = proyecto;

break;

}

}

if (proyectoSeleccionado == null) {

System.out.println("Proyecto no encontrado.");

return;

}

System.out.println("\n--- Características del Proyecto ---");

System.out.println("Nombre: " + proyectoSeleccionado.getNombre());

System.out.println("Descripción: " + proyectoSeleccionado.getDescripcion());

System.out.println("Fecha de Inicio: " + proyectoSeleccionado.getFechaInicio());

System.out.println("Fecha de Finalización: " + proyectoSeleccionado.getFechaFin());

System.out.println("Estado: " + proyectoSeleccionado.getEstado());

System.out.println("Prioridad: " + proyectoSeleccionado.getPrioridad());

System.out.println("\n--- Miembros Asignados ---");

for (Miembro miembro : proyectoSeleccionado.getMiembros()) {

System.out.println("Miembro: " + miembro.getNombre() + " - Rol: " + miembro.getRol());

}

System.out.println("\n--- Tareas Asignadas ---");

for (Tarea tarea : proyectoSeleccionado.getTareas()) {

System.out.println("Tarea: " + tarea.getNombre() + " - Estado: " + tarea.getEstado());

}

System.out.println("\n--- Reportes de Desempeño ---");

for (ReporteDesempeño reporte : proyectoSeleccionado.getReportes()) {

System.out.println(reporte);

}

while (true) {

System.out.println("\n--- Opciones ---");

System.out.println("1. Agregar miembro");

System.out.println("2. Eliminar miembro");

System.out.println("3. Agregar tarea");

System.out.println("4. Eliminar tarea");

System.out.println("5. Volver al menú principal");

System.out.print("Seleccione una opción: ");

int opcion = scanner.nextInt();

scanner.nextLine();

switch (opcion) {

case 1:

agregarMiembroAProyecto(scanner, proyectoSeleccionado, usuarios);

break;

case 2:

eliminarMiembroDeProyecto(scanner, proyectoSeleccionado);

break;

case 3:

agregarTareaAProyecto(scanner, proyectoSeleccionado);

break;

case 4:

eliminarTareaDeProyecto(scanner, proyectoSeleccionado);

break;

case 5:

return;

default:

System.out.println("Opción no válida.");

}

}

}

private void agregarMiembroAProyecto(Scanner scanner, Proyecto proyecto, List<Usuario> usuarios) {

System.out.print("Nombre del miembro a agregar: ");

String nombreMiembro = scanner.nextLine();

System.out.print("Rol del miembro: ");

String rolMiembro = scanner.nextLine();

Miembro miembro = new Miembro(nombreMiembro, rolMiembro);

proyecto.asignarMiembros(Collections.singletonList(miembro));

System.out.println("Miembro agregado al proyecto: " + nombreMiembro);

}

private void eliminarMiembroDeProyecto(Scanner scanner, Proyecto proyecto) {

System.out.print("Nombre del miembro a eliminar: ");

String nombreMiembro = scanner.nextLine();

proyecto.getMiembros().removeIf(miembro -> miembro.getNombre().equals(nombreMiembro));

System.out.println("Miembro eliminado del proyecto: " + nombreMiembro);

}

private void agregarTareaAProyecto(Scanner scanner, Proyecto proyecto) {

System.out.println("\n--- Tareas Disponibles ---");

for (Tarea tarea : tareas) {

System.out.println("- " + tarea.getNombre());

}

System.out.print("Seleccione el nombre de la tarea para agregar al proyecto: ");

String nombreTarea = scanner.nextLine();

Tarea tareaSeleccionada = null;

for (Tarea tarea : tareas) {

if (tarea.getNombre().equals(nombreTarea)) {

tareaSeleccionada = tarea;

break;

}

}

if (tareaSeleccionada == null) {

System.out.println("Tarea no encontrada.");

return;

}

proyecto.asignarTareas(Collections.singletonList(tareaSeleccionada));

System.out.println("Tarea " + tareaSeleccionada.getNombre() + " agregada al proyecto " + proyecto.getNombre());

}

private void eliminarTareaDeProyecto(Scanner scanner, Proyecto proyecto) {

System.out.println("\n--- Tareas Asignadas ---");

for (int i = 0; i < proyecto.getTareas().size(); i++) {

System.out.println((i + 1) + ". " + proyecto.getTareas().get(i).getNombre());

}

System.out.print("Seleccione el número de la tarea a eliminar: ");

int opcion = scanner.nextInt();

scanner.nextLine();

if (opcion < 1 || opcion > proyecto.getTareas().size()) {

System.out.println("Opción no válida.");

return;

}

Tarea tareaSeleccionada = proyecto.getTareas().get(opcion - 1);

proyecto.getTareas().remove(tareaSeleccionada);

System.out.println("Tarea eliminada del proyecto: " + tareaSeleccionada.getNombre());

}

public void editarProyecto(Scanner scanner) {

System.out.println("\n--- Proyectos Disponibles ---");

for (Proyecto proyecto : proyectos) {

System.out.println("- " + proyecto.getNombre());

}

System.out.print("Seleccione el nombre del proyecto a editar: ");

String nombreProyecto = scanner.nextLine();

Proyecto proyectoSeleccionado = null;

for (Proyecto proyecto : proyectos) {

if (proyecto.getNombre().equals(nombreProyecto)) {

proyectoSeleccionado = proyecto;

break;

}

}

if (proyectoSeleccionado == null) {

System.out.println("Proyecto no encontrado.");

return;

}

System.out.println("\n--- Atributos del Proyecto ---");

System.out.println("1. Nombre");

System.out.println("2. Descripción");

System.out.println("3. Fecha de inicio");

System.out.println("4. Fecha de finalización");

System.out.println("5. Estado");

System.out.println("6. Prioridad");

System.out.print("Seleccione el número del atributo que desea editar: ");

int opcion = scanner.nextInt();

scanner.nextLine();

switch (opcion) {

case 1:

System.out.print("Nuevo nombre: ");

String nuevoNombre = scanner.nextLine();

proyectoSeleccionado.setNombre(nuevoNombre);

System.out.println("Nombre actualizado.");

break;

case 2:

System.out.print("Nueva descripción: ");

String nuevaDescripcion = scanner.nextLine();

proyectoSeleccionado.setDescripcion(nuevaDescripcion);

System.out.println("Descripción actualizada.");

break;

case 3:

System.out.print("Nueva fecha de inicio (yyyy-MM-dd): ");

String nuevaFechaInicioStr = scanner.nextLine();

Date nuevaFechaInicio = parseFecha(nuevaFechaInicioStr);

proyectoSeleccionado.setFechaInicio(nuevaFechaInicio);

System.out.println("Fecha de inicio actualizada.");

break;

case 4:

System.out.print("Nueva fecha de finalización (yyyy-MM-dd): ");

String nuevaFechaFinStr = scanner.nextLine();

Date nuevaFechaFin = parseFecha(nuevaFechaFinStr);

proyectoSeleccionado.setFechaFin(nuevaFechaFin);

System.out.println("Fecha de finalización actualizada.");

break;

case 5:

System.out.print("Nuevo estado: ");

String nuevoEstado = scanner.nextLine();

proyectoSeleccionado.setEstado(nuevoEstado);

System.out.println("Estado actualizado.");

break;

case 6:

System.out.print("Nueva prioridad: ");

String nuevaPrioridad = scanner.nextLine();

proyectoSeleccionado.setPrioridad(nuevaPrioridad);

System.out.println("Prioridad actualizada.");

break;

default:

System.out.println("Opción no válida.");

}

}

public void eliminarProyecto(Scanner scanner) {

System.out.println("\n--- Proyectos Disponibles ---");

for (int i = 0; i < proyectos.size(); i++) {

System.out.println((i + 1) + ". " + proyectos.get(i).getNombre());

}

System.out.print("Seleccione el número del proyecto a eliminar: ");

int opcion = scanner.nextInt();

scanner.nextLine();

if (opcion < 1 || opcion > proyectos.size()) {

System.out.println("Opción no válida.");

return;

}

Proyecto proyectoSeleccionado = proyectos.get(opcion - 1);

proyectos.remove(proyectoSeleccionado);

System.out.println("Proyecto eliminado: " + proyectoSeleccionado.getNombre());

}

public void asignarMiembrosAProyecto(String nombreProyecto, List<Miembro> miembros, String asignador) {

for (Proyecto proyecto : proyectos) {

if (proyecto.getNombre().equals(nombreProyecto)) {

for (Miembro miembro : miembros) {

miembro.setAsignador(asignador);

}

proyecto.asignarMiembros(miembros);

return;

}

}

System.out.println("Proyecto no encontrado.");

}

public void asignarMiembrosAProyecto(Scanner scanner, List<Usuario> usuarios, String usuarioActual, Proyecto proyecto) {

System.out.println("Miembros disponibles:");

for (Usuario usuario : usuarios) {

if (!usuario.getNombre().equals(usuarioActual)) {

System.out.println("- " + usuario.getNombre());

}

}

System.out.print("Seleccione el nombre del miembro a asignar: ");

String nombreMiembro = scanner.nextLine();

System.out.print("Rol del miembro: ");

String rolMiembro = scanner.nextLine();

Miembro miembro = new Miembro(nombreMiembro, rolMiembro);

proyecto.asignarMiembros(Collections.singletonList(miembro));

miembro.setAsignador(usuarioActual);

System.out.println("Miembro asignado al proyecto: " + proyecto.getNombre());

}

public void crearTarea(String nombre, String descripcion, Date fechaCreacion, Date fechaLimite, String prioridad, String estado) {

Tarea tarea = new Tarea(nombre, descripcion, fechaCreacion, fechaLimite, prioridad, estado);

tareas.add(tarea);

}

public void crearTarea(Scanner scanner) {

System.out.print("Nombre de la tarea: ");

String nombre = scanner.nextLine();

System.out.print("Descripción: ");

String descripcion = scanner.nextLine();

System.out.print("Fecha límite (yyyy-MM-dd): ");

String fechaLimiteStr = scanner.nextLine();

Date fechaLimite = parseFecha(fechaLimiteStr);

System.out.print("Prioridad: ");

String prioridad = scanner.nextLine();

System.out.print("Estado: ");

String estado = scanner.nextLine();

Tarea tarea = new Tarea(nombre, descripcion, new Date(), fechaLimite, prioridad, estado);

tareas.add(tarea);

System.out.println("Tarea creada: " + nombre);

}

public void verTareas(Scanner scanner) {

System.out.println("\n--- Tareas Disponibles ---");

for (Tarea tarea : tareas) {

System.out.println("- " + tarea.getNombre());

}

System.out.print("Seleccione el nombre de la tarea para ver sus características: ");

String nombreTarea = scanner.nextLine();

Tarea tareaSeleccionada = null;

for (Tarea tarea : tareas) {

if (tarea.getNombre().equals(nombreTarea)) {

tareaSeleccionada = tarea;

break;

}

}

if (tareaSeleccionada == null) {

System.out.println("Tarea no encontrada.");

return;

}

System.out.println("\n--- Características de la Tarea ---");

System.out.println("Nombre: " + tareaSeleccionada.getNombre());

System.out.println("Descripción: " + tareaSeleccionada.getDescripcion());

System.out.println("Fecha de Creación: " + tareaSeleccionada.getFechaCreacion());

System.out.println("Fecha Límite: " + tareaSeleccionada.getFechaLimite());

System.out.println("Prioridad: " + tareaSeleccionada.getPrioridad());

System.out.println("Estado: " + tareaSeleccionada.getEstado());

System.out.println("\n--- Miembros Asignados ---");

for (Miembro miembro : tareaSeleccionada.getMiembrosAsignados()) {

System.out.println("Miembro: " + miembro.getNombre() + " - Rol: " + miembro.getRol());

}

System.out.println("\n--- Comentarios ---");

for (Comentario comentario : tareaSeleccionada.getComentarios()) {

System.out.println("Autor: " + comentario.getAutor().getNombre() + " - Comentario: " + comentario.getTexto());

}

while (true) {

System.out.println("\n--- Opciones ---");

System.out.println("1. Editar características de la tarea");

System.out.println("2. Agregar miembro a la tarea");

System.out.println("3. Eliminar miembro de la tarea");

System.out.println("4. Agregar comentario a la tarea");

System.out.println("5. Volver al menú principal");

System.out.print("Seleccione una opción: ");

int opcion = scanner.nextInt();

scanner.nextLine();

switch (opcion) {

case 1:

editarTarea(scanner, tareaSeleccionada);

break;

case 2:

agregarMiembroATarea(scanner, tareaSeleccionada);

break;

case 3:

eliminarMiembroDeTarea(scanner, tareaSeleccionada);

break;

case 4:

agregarComentarioATarea(scanner, tareaSeleccionada);

break;

case 5:

return;

default:

System.out.println("Opción no válida.");

}

}

}

private void editarTarea(Scanner scanner, Tarea tarea) {

System.out.println("\n--- Atributos de la Tarea ---");

System.out.println("1. Nombre");

System.out.println("2. Descripción");

System.out.println("3. Fecha de Creación");

System.out.println("4. Fecha Límite");

System.out.println("5. Prioridad");

System.out.println("6. Estado");

System.out.print("Seleccione el número del atributo que desea editar: ");

int opcion = scanner.nextInt();

scanner.nextLine();

switch (opcion) {

case 1:

System.out.print("Nuevo nombre: ");

String nuevoNombre = scanner.nextLine();

tarea.setNombre(nuevoNombre);

System.out.println("Nombre actualizado.");

break;

case 2:

System.out.print("Nueva descripción: ");

String nuevaDescripcion = scanner.nextLine();

tarea.setDescripcion(nuevaDescripcion);

System.out.println("Descripción actualizada.");

break;

case 3:

System.out.print("Nueva fecha de creación (yyyy-MM-dd): ");

String nuevaFechaCreacionStr = scanner.nextLine();

Date nuevaFechaCreacion = parseFecha(nuevaFechaCreacionStr);

tarea.setFechaCreacion(nuevaFechaCreacion);

System.out.println("Fecha de creación actualizada.");

break;

case 4:

System.out.print("Nueva fecha límite (yyyy-MM-dd): ");

String nuevaFechaLimiteStr = scanner.nextLine();

Date nuevaFechaLimite = parseFecha(nuevaFechaLimiteStr);

tarea.setFechaLimite(nuevaFechaLimite);

System.out.println("Fecha límite actualizada.");

break;

case 5:

System.out.print("Nueva prioridad: ");

String nuevaPrioridad = scanner.nextLine();

tarea.setPrioridad(nuevaPrioridad);

System.out.println("Prioridad actualizada.");

break;

case 6:

System.out.print("Nuevo estado: ");

String nuevoEstado = scanner.nextLine();

tarea.setEstado(nuevoEstado);

System.out.println("Estado actualizado.");

break;

default:

System.out.println("Opción no válida.");

}

}

private void agregarMiembroATarea(Scanner scanner, Tarea tarea) {

System.out.print("Nombre del miembro a agregar: ");

String nombreMiembro = scanner.nextLine();

Miembro miembro = new Miembro(nombreMiembro, "Rol");

tarea.asignarMiembro(miembro);

System.out.println("Miembro agregado a la tarea: " + nombreMiembro);

}

private void eliminarMiembroDeTarea(Scanner scanner, Tarea tarea) {

System.out.print("Nombre del miembro a eliminar: ");

String nombreMiembro = scanner.nextLine();

boolean encontrado = false;

for (Miembro miembro : tarea.getMiembrosAsignados()) {

if (miembro.getNombre().equals(nombreMiembro)) {

tarea.getMiembrosAsignados().remove(miembro);

System.out.println("Miembro eliminado de la tarea: " + nombreMiembro);

encontrado = true;

break;

}

}

if (!encontrado) {

System.out.println("Miembro no encontrado en la tarea.");

}

}

private void agregarComentarioATarea(Scanner scanner, Tarea tarea) {

System.out.print("Comentario: ");

String comentario = scanner.nextLine();

System.out.print("Nombre del autor: ");

String nombreAutor = scanner.nextLine();

Miembro autor = new Miembro(nombreAutor, "");

Comentario nuevoComentario = new Comentario(comentario, new Date(), autor);

tarea.agregarComentario(nuevoComentario);

System.out.println("Comentario agregado a la tarea.");

}

public void asignarTareaAMiembro(Scanner scanner, List<Usuario> usuarios, Usuario usuarioActual) {

System.out.println("\n--- Tareas Disponibles ---");

for (Tarea tarea : tareas) {

System.out.println("- " + tarea.getNombre());

}

System.out.print("Seleccione el nombre de la tarea: ");

String nombreTarea = scanner.nextLine();

Tarea tareaSeleccionada = null;

for (Tarea tarea : tareas) {

if (tarea.getNombre().equals(nombreTarea)) {

tareaSeleccionada = tarea;

break;

}

}

if (tareaSeleccionada == null) {

System.out.println("Tarea no encontrada.");

return;

}

System.out.println("Miembros disponibles:");

for (Usuario usuario : usuarios) {

if (!usuario.getNombre().equals(usuarioActual.getNombre())) {

System.out.println("- " + usuario.getNombre());

}

}

System.out.print("Seleccione el nombre del miembro: ");

String nombreMiembro = scanner.nextLine();

for (Usuario usuario : usuarios) {

if (usuario.getNombre().equals(nombreMiembro)) {

Miembro miembro = new Miembro(usuario.getNombre(), "Rol");

tareaSeleccionada.asignarMiembro(miembro);

System.out.println("Tarea asignada a miembro: " + nombreTarea + " -> " + nombreMiembro);

return;

}

}

System.out.println("Usuario no encontrado.");

}

public void actualizarEstadoTarea(Scanner scanner) {

System.out.println("\n--- Tareas Disponibles ---");

for (Tarea tarea : tareas) {

System.out.println("- " + tarea.getNombre());

}

System.out.print("Seleccione el nombre de la tarea para actualizar su estado: ");

String nombreTarea = scanner.nextLine();

Tarea tareaSeleccionada = null;

for (Tarea tarea : tareas) {

if (tarea.getNombre().equals(nombreTarea)) {

tareaSeleccionada = tarea;

break;

}

}

if (tareaSeleccionada == null) {

System.out.println("Tarea no encontrada.");

return;

}

System.out.print("Nuevo estado: ");

String nuevoEstado = scanner.nextLine();

tareaSeleccionada.setEstado(nuevoEstado);

System.out.println("Estado de la tarea actualizado: " + nombreTarea + " -> " + nuevoEstado);

}

public void agregarComentarioATarea(Scanner scanner) {

System.out.print("Nombre de la tarea: ");

String nombreTarea = scanner.nextLine();

System.out.print("Comentario: ");

String comentario = scanner.nextLine();

System.out.print("Nombre del autor: ");

String nombreAutor = scanner.nextLine();

Miembro autor = new Miembro(nombreAutor, "");

Comentario nuevoComentario = new Comentario(comentario, new Date(), autor);

for (Tarea tarea : tareas) {

if (tarea.getNombre().equals(nombreTarea)) {

tarea.agregarComentario(nuevoComentario);

System.out.println("Comentario agregado a la tarea: " + nombreTarea);

return;

}

}

System.out.println("Tarea no encontrada.");

}

public void generarInformeProgreso(Scanner scanner) {

System.out.print("Nombre del proyecto: ");

String nombreProyecto = scanner.nextLine();

Proyecto proyectoSeleccionado = null;

for (Proyecto proyecto : proyectos) {

if (proyecto.getNombre().equals(nombreProyecto)) {

proyectoSeleccionado = proyecto;

break;

}

}

if (proyectoSeleccionado == null) {

System.out.println("Proyecto no encontrado.");

return;

}

int tareasCompletadas = 0;

int tareasPendientes = 0;

for (Tarea tarea : proyectoSeleccionado.getTareas()) {

if (tarea.getEstado().equalsIgnoreCase("Completada") ||

tarea.getEstado().equalsIgnoreCase("Terminada") ||

tarea.getEstado().equalsIgnoreCase("Entregada")) {

tareasCompletadas++;

} else {

tareasPendientes++;

}

}

float porcentajeCompletado = 0;

if (tareasCompletadas + tareasPendientes > 0) {

porcentajeCompletado = (float) tareasCompletadas / (tareasCompletadas + tareasPendientes) \* 100;

}

InformeDeProgreso informe = new InformeDeProgreso("Progreso del proyecto: " + nombreProyecto, tareasCompletadas, tareasPendientes, porcentajeCompletado);

System.out.println(informe.verInforme());

}

public void enviarMensaje(Scanner scanner, List<Usuario> usuarios, Usuario remitente) {

System.out.println("\n--- Enviar Mensaje ---");

System.out.print("Seleccione el destinatario: ");

System.out.println();

for (Usuario usuario : usuarios) {

if (!usuario.getNombre().equals(remitente.getNombre())) {

System.out.println("- " + usuario.getNombre());

}

}

String nombreDestinatario = scanner.nextLine();

System.out.print("Mensaje: ");

String mensaje = scanner.nextLine();

Mensaje nuevoMensaje = new Mensaje(mensaje, new Date(), new Miembro(nombreDestinatario, ""));

mensajes.add(nuevoMensaje);

System.out.println("Mensaje enviado a: " + nombreDestinatario);

for (Usuario usuario : usuarios) {

if (usuario.getNombre().equals(nombreDestinatario)) {

usuario.agregarMensaje(nuevoMensaje);

nuevoMensaje.getDestinatario().setNombre(remitente.getNombre());

break;

}

}

}

public void verMensajesRecibidos(Usuario usuario) {

System.out.println("\n--- Mensajes Recibidos ---");

for (Mensaje mensaje : usuario.getMensajesRecibidos()) {

System.out.println("De: " + mensaje.getDestinatario().getNombre() + " - Mensaje: " + mensaje.getTexto());

}

}

public void crearEquipo(Scanner scanner, List<Usuario> usuarios) {

System.out.print("Nombre del equipo: ");

String nombreEquipo = scanner.nextLine();

Equipo equipo = new Equipo(nombreEquipo);

while (true) {

System.out.println("\n--- Miembros Disponibles ---");

for (Usuario usuario : usuarios) {

System.out.println("- " + usuario.getNombre());

}

System.out.print("Seleccione el nombre del miembro a agregar al equipo (o escriba 'fin' para terminar): ");

String nombreMiembro = scanner.nextLine();

if (nombreMiembro.equalsIgnoreCase("fin")) {

break;

}

boolean miembroEncontrado = false;

for (Usuario usuario : usuarios) {

if (usuario.getNombre().equals(nombreMiembro)) {

System.out.print("Rol del miembro: ");

String rolMiembro = scanner.nextLine();

Miembro miembro = new Miembro(usuario.getNombre(), rolMiembro);

equipo.agregarMiembro(miembro);

miembroEncontrado = true;

System.out.println("Miembro agregado: " + nombreMiembro + " con rol: " + rolMiembro);

break;

}

}

if (!miembroEncontrado) {

System.out.println("Miembro no encontrado. Intente de nuevo.");

}

System.out.print("¿Desea agregar más miembros? (s/n): ");

String respuesta = scanner.nextLine();

if (respuesta.equalsIgnoreCase("n")) {

break;

}

}

equipos.add(equipo);

System.out.println("Equipo creado: " + nombreEquipo);

}

public void verEquipos(Scanner scanner) {

while (true) {

System.out.println("\n--- Equipos Disponibles ---");

for (int i = 0; i < equipos.size(); i++) {

System.out.println((i + 1) + ". " + equipos.get(i).getNombre());

}

System.out.print("Seleccione el número del equipo para ver sus detalles (o 0 para volver): ");

int opcion = scanner.nextInt();

scanner.nextLine();

if (opcion == 0) {

return;

}

if (opcion < 1 || opcion > equipos.size()) {

System.out.println("Opción no válida.");

continue;

}

Equipo equipoSeleccionado = equipos.get(opcion - 1);

System.out.println("\n--- Detalles del Equipo: " + equipoSeleccionado.getNombre() + " ---");

System.out.println("Fecha de Creación: " + equipoSeleccionado.getFechaCreacion());

System.out.println("--- Miembros ---");

for (Miembro miembro : equipoSeleccionado.getMiembros()) {

System.out.println("Miembro: " + miembro.getNombre() + " - Rol: " + miembro.getRol());

}

System.out.println("1. Editar nombre del equipo");

System.out.println("2. Agregar miembro");

System.out.println("3. Quitar miembro");

System.out.println("4. Eliminar este equipo");

System.out.println("5. Regresar a la lista de equipos");

System.out.print("Seleccione una opción: ");

int opcionSeleccionada = scanner.nextInt();

scanner.nextLine();

switch (opcionSeleccionada) {

case 1:

System.out.print("Nuevo nombre del equipo: ");

String nuevoNombre = scanner.nextLine();

equipoSeleccionado.setNombre(nuevoNombre);

System.out.println("Nombre del equipo actualizado a: " + nuevoNombre);

break;

case 2:

System.out.print("Nombre del miembro a agregar: ");

String nombreMiembroAgregar = scanner.nextLine();

System.out.print("Rol del miembro: ");

String rolMiembroAgregar = scanner.nextLine();

Miembro nuevoMiembro = new Miembro(nombreMiembroAgregar, rolMiembroAgregar);

equipoSeleccionado.agregarMiembro(nuevoMiembro);

System.out.println("Miembro agregado: " + nombreMiembroAgregar + " con rol: " + rolMiembroAgregar);

break;

case 3:

System.out.print("Nombre del miembro a quitar: ");

String nombreMiembroQuitar = scanner.nextLine();

boolean encontrado = false;

for (Miembro miembro : equipoSeleccionado.getMiembros()) {

if (miembro.getNombre().equals(nombreMiembroQuitar)) {

equipoSeleccionado.getMiembros().remove(miembro);

System.out.println("Miembro quitado: " + nombreMiembroQuitar);

encontrado = true;

break;

}

}

if (!encontrado) {

System.out.println("Miembro no encontrado en el equipo.");

}

break;

case 4:

equipos.remove(equipoSeleccionado);

System.out.println("Equipo eliminado: " + equipoSeleccionado.getNombre());

return;

case 5:

continue;

default:

System.out.println("Opción no válida.");

}

}

}

public void asignarPermisos(Scanner scanner) {

System.out.print("Nombre del usuario: ");

String nombreUsuario = scanner.nextLine();

System.out.print("Nombre del proyecto: ");

String nombreProyecto = scanner.nextLine();

System.out.print("Nivel de acceso (Lectura/Escritura/Administrador): ");

String nivelAcceso = scanner.nextLine();

PermisosAcceso permisos = new PermisosAcceso(new Miembro(nombreUsuario, ""), new Proyecto(nombreProyecto, "", new Date(), new Date(), "", "", false), nivelAcceso);

System.out.println("Permisos asignados: " + nombreUsuario + " -> " + nombreProyecto);

}

public void generarReporteDesempeño(Scanner scanner) {

System.out.print("Nombre del miembro: ");

String nombreMiembro = scanner.nextLine();

System.out.print("Nombre del proyecto: ");

String nombreProyecto = scanner.nextLine();

Proyecto proyectoSeleccionado = null;

for (Proyecto proyecto : proyectos) {

if (proyecto.getNombre().equals(nombreProyecto)) {

proyectoSeleccionado = proyecto;

break;

}

}

if (proyectoSeleccionado == null) {

System.out.println("Proyecto no encontrado.");

return;

}

int tareasCompletadas = 0;

int tareasPendientes = 0;

for (Tarea tarea : proyectoSeleccionado.getTareas()) {

if (tarea.getEstado().equalsIgnoreCase("Completada") ||

tarea.getEstado().equalsIgnoreCase("Terminada") ||

tarea.getEstado().equalsIgnoreCase("Entregada")) {

tareasCompletadas++;

} else {

tareasPendientes++;

}

}

float porcentajeCompletado = 0;

if (tareasCompletadas + tareasPendientes > 0) {

porcentajeCompletado = (float) tareasCompletadas / (tareasCompletadas + tareasPendientes) \* 100;

}

InformeDeProgreso informe = new InformeDeProgreso("Progreso del proyecto: " + nombreProyecto, tareasCompletadas, tareasPendientes, porcentajeCompletado);

proyectoSeleccionado.agregarReporte(new ReporteDesempeño(nombreMiembro, nombreProyecto, new Date(), informe.verInforme()));

System.out.println("Reporte de desempeño generado para: " + nombreMiembro + " en " + nombreProyecto);

}

public void verTareasYProyectosAsignados(Usuario usuario) {

System.out.println("\n--- Tareas y Proyectos Asignados a " + usuario.getNombre() + " ---");

boolean hayAsignaciones = false;

for (Proyecto proyecto : proyectos) {

for (Miembro miembro : proyecto.getMiembros()) {

if (miembro.getNombre().equals(usuario.getNombre())) {

System.out.println("Proyecto: " + proyecto.getNombre() + " - Rol: " + miembro.getRol() + " - Asignado por: " + miembro.getAsignador());

hayAsignaciones = true;

for (Tarea tarea : tareas) {

if (tarea.getMiembrosAsignados().contains(miembro)) {

System.out.println(" Tarea: " + tarea.getNombre());

}

}

}

}

}

if (!hayAsignaciones) {

System.out.println("No hay proyectos o tareas asignados a " + usuario.getNombre());

}

}

public List<Proyecto> getProyectos() {

return proyectos;

}

public List<Tarea> getTareas() {

return tareas;

}

public List<Equipo> getEquipos() {

return equipos;

}

public boolean eliminarTarea(String nombreTarea) {

return tareas.removeIf(tarea -> tarea.getNombre().equals(nombreTarea));

}

public boolean eliminarEquipo(String nombreEquipo) {

return equipos.removeIf(equipo -> equipo.getNombre().equals(nombreEquipo));

}

public void eliminarMiembroDeProyectos(String nombreMiembro) {

for (Proyecto proyecto : proyectos) {

proyecto.getMiembros().removeIf(miembro -> miembro.getNombre().equals(nombreMiembro));

for (Tarea tarea : proyecto.getTareas()) {

tarea.getMiembrosAsignados().removeIf(miembro -> miembro.getNombre().equals(nombreMiembro));

}

}

for (Equipo equipo : equipos) {

equipo.getMiembros().removeIf(miembro -> miembro.getNombre().equals(nombreMiembro));

}

}

}

class Usuario {

private String nombre;

private String contrasena;

private String nivelPermisos;

private List<Mensaje> mensajesRecibidos = new ArrayList<>();

public Usuario(String nombre, String contrasena, String nivelPermisos) {

this.nombre = nombre;

this.contrasena = contrasena;

this.nivelPermisos = nivelPermisos;

}

public String getNombre() {

return nombre;

}

public String getContrasena() {

return contrasena;

}

public String getNivelPermisos() {

return nivelPermisos;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public void setContrasena(String contrasena) {

this.contrasena = contrasena;

}

public void setNivelPermisos(String nivelPermisos) {

this.nivelPermisos = nivelPermisos;

}

public void agregarMensaje(Mensaje mensaje) {

mensajesRecibidos.add(mensaje);

}

public List<Mensaje> getMensajesRecibidos() {

return mensajesRecibidos;

}

public List<Tarea> getTareasAsignadas(GestorProyectos gestorProyectos) {

List<Tarea> tareasAsignadas = new ArrayList<>();

for (Proyecto proyecto : gestorProyectos.getProyectos()) {

for (Tarea tarea : proyecto.getTareas()) {

for (Miembro miembro : tarea.getMiembrosAsignados()) {

if (miembro.getNombre().equals(this.nombre)) {

tareasAsignadas.add(tarea);

}

}

}

}

return tareasAsignadas;

}

public List<Equipo> getEquiposAsignados(GestorProyectos gestorProyectos) {

List<Equipo> equiposAsignados = new ArrayList<>();

for (Equipo equipo : gestorProyectos.getEquipos()) {

for (Miembro miembro : equipo.getMiembros()) {

if (miembro.getNombre().equals(this.nombre)) {

equiposAsignados.add(equipo);

}

}

}

return equiposAsignados;

}

public List<Proyecto> getProyectosAsignados(GestorProyectos gestorProyectos) {

List<Proyecto> proyectosAsignados = new ArrayList<>();

for (Proyecto proyecto : gestorProyectos.getProyectos()) {

for (Miembro miembro : proyecto.getMiembros()) {

if (miembro.getNombre().equals(this.nombre)) {

proyectosAsignados.add(proyecto);

}

}

}

return proyectosAsignados;

}

}

class Proyecto {

private String nombre;

private String descripcion;

private Date fechaInicio;

private Date fechaFin;

private String estado;

private String prioridad;

private List<Miembro> miembros;

private List<Tarea> tareas;

private List<ReporteDesempeño> reportes;

public Proyecto(String nombre, String descripcion, Date fechaInicio, Date fechaFin, String estado, String prioridad, boolean preregistrado) {

this.nombre = nombre;

this.descripcion = descripcion;

this.fechaInicio = fechaInicio;

this.fechaFin = fechaFin;

this.estado = estado;

this.prioridad = prioridad;

this.miembros = new ArrayList<>();

this.tareas = new ArrayList<>();

this.reportes = new ArrayList<>();

if (preregistrado) {

if (fechaInicio.after(new Date(System.currentTimeMillis() - 3L \* 24 \* 60 \* 60 \* 1000))) {

throw new IllegalArgumentException("La fecha de inicio debe ser de al menos hace 3 días.");

}

if (fechaFin.before(new Date())) {

throw new IllegalArgumentException("La fecha de finalización debe ser futura.");

}

}

}

public String getNombre() {

return nombre;

}

public String getDescripcion() {

return descripcion;

}

public Date getFechaInicio() {

return fechaInicio;

}

public Date getFechaFin() {

return fechaFin;

}

public String getEstado() {

return estado;

}

public String getPrioridad() {

return prioridad;

}

public List<Miembro> getMiembros() {

return miembros;

}

public List<Tarea> getTareas() {

return tareas;

}

public List<ReporteDesempeño> getReportes() {

return reportes;

}

public void asignarMiembros(List<Miembro> miembros) {

this.miembros.addAll(miembros);

}

public void asignarTareas(List<Tarea> tareas) {

this.tareas.addAll(tareas);

}

public void agregarReporte(ReporteDesempeño reporte) {

reportes.add(reporte);

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public void setDescripcion(String descripcion) {

this.descripcion = descripcion;

}

public void setFechaInicio(Date fechaInicio) {

this.fechaInicio = fechaInicio;

}

public void setFechaFin(Date fechaFin) {

this.fechaFin = fechaFin;

}

public void setEstado(String estado) {

this.estado = estado;

}

public void setPrioridad(String prioridad) {

this.prioridad = prioridad;

}

}

class Miembro {

private String nombre;

private String rol;

private String nivelPermisos;

private String asignador;

public Miembro(String nombre, String rol) {

this.nombre = nombre;

this.rol = rol;

this.nivelPermisos = "Usuario";

}

public String getNombre() {

return nombre;

}

public String getRol() {

return rol;

}

public String getNivelPermisos() {

return nivelPermisos;

}

public void setNivelPermisos(String nivelPermisos) {

this.nivelPermisos = nivelPermisos;

}

public String getAsignador() {

return asignador;

}

public void setAsignador(String asignador) {

this.asignador = asignador;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

}

class Tarea {

private String nombre;

private String descripcion;

private Date fechaCreacion;

private Date fechaLimite;

private String prioridad;

private String estado;

private List<Miembro> miembrosAsignados = new ArrayList<>();

private List<Comentario> comentarios = new ArrayList<>();

public Tarea(String nombre, String descripcion, Date fechaCreacion, Date fechaLimite, String prioridad, String estado) {

this.nombre = nombre;

this.descripcion = descripcion;

this.fechaCreacion = fechaCreacion;

this.fechaLimite = fechaLimite;

this.prioridad = prioridad;

this.estado = estado;

}

public String getNombre() {

return nombre;

}

public String getDescripcion() {

return descripcion;

}

public Date getFechaCreacion() {

return fechaCreacion;

}

public Date getFechaLimite() {

return fechaLimite;

}

public String getPrioridad() {

return prioridad;

}

public String getEstado() {

return estado;

}

public List<Miembro> getMiembrosAsignados() {

return miembrosAsignados;

}

public void asignarMiembro(Miembro miembro) {

miembrosAsignados.add(miembro);

}

public void agregarComentario(Comentario comentario) {

comentarios.add(comentario);

}

public List<Comentario> getComentarios() {

return comentarios;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public void setDescripcion(String descripcion) {

this.descripcion = descripcion;

}

public void setFechaCreacion(Date fechaCreacion) {

this.fechaCreacion = fechaCreacion;

}

public void setFechaLimite(Date fechaLimite) {

this.fechaLimite = fechaLimite;

}

public void setPrioridad(String prioridad) {

this.prioridad = prioridad;

}

public void setEstado(String estado) {

this.estado = estado;

}

}

class Comentario {

private String texto;

private Date fecha;

private Miembro autor;

public Comentario(String texto, Date fecha, Miembro autor) {

this.texto = texto;

this.fecha = fecha;

this.autor = autor;

}

public String getTexto() {

return texto;

}

public Date getFecha() {

return fecha;

}

public Miembro getAutor() {

return autor;

}

}

class Mensaje {

private String texto;

private Date fecha;

private Miembro destinatario;

public Mensaje(String texto, Date fecha, Miembro destinatario) {

this.texto = texto;

this.fecha = fecha;

this.destinatario = destinatario;

}

public String getTexto() {

return texto;

}

public Date getFecha() {

return fecha;

}

public Miembro getDestinatario() {

return destinatario;

}

}

class Equipo {

private String nombre;

private List<Miembro> miembros = new ArrayList<>();

private Date fechaCreacion;

public Equipo(String nombre) {

this.nombre = nombre;

this.fechaCreacion = new Date();

}

public void agregarMiembro(Miembro miembro) {

miembros.add(miembro);

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public List<Miembro> getMiembros() {

return miembros;

}

public Date getFechaCreacion() {

return fechaCreacion;

}

}

class PermisosAcceso {

private Miembro miembro;

private Proyecto proyecto;

private String nivelAcceso;

public PermisosAcceso(Miembro miembro, Proyecto proyecto, String nivelAcceso) {

this.miembro = miembro;

this.proyecto = proyecto;

this.nivelAcceso = nivelAcceso;

}

public Miembro getMiembro() {

return miembro;

}

public Proyecto getProyecto() {

return proyecto;

}

public String getNivelAcceso() {

return nivelAcceso;

}

}

class InformeDeProgreso {

private String descripcion;

private int tareasCompletadas;

private int tareasPendientes;

private float porcentajeCompletado;

public InformeDeProgreso(String descripcion, int tareasCompletadas, int tareasPendientes, float porcentajeCompletado) {

this.descripcion = descripcion;

this.tareasCompletadas = tareasCompletadas;

this.tareasPendientes = tareasPendientes;

this.porcentajeCompletado = porcentajeCompletado;

}

public String verInforme() {

return "Informe: " + descripcion + "\nTareas Completadas: " + tareasCompletadas + "\nTareas Pendientes: " + tareasPendientes + "\nPorcentaje Completado: " + porcentajeCompletado + "%";

}

}

class ReporteDesempeño {

private String nombreMiembro;

private String nombreProyecto;

private Date fechaGeneracion;

private String contenido;

public ReporteDesempeño(String nombreMiembro, String nombreProyecto, Date fechaGeneracion, String contenido) {

this.nombreMiembro = nombreMiembro;

this.nombreProyecto = nombreProyecto;

this.fechaGeneracion = fechaGeneracion;

this.contenido = contenido;

}

public String getNombreMiembro() {

return nombreMiembro;

}

public String getNombreProyecto() {

return nombreProyecto;

}

public Date getFechaGeneracion() {

return fechaGeneracion;

}

public String getContenido() {

return contenido;

}

@Override

public String toString() {

return "Reporte de Desempeño para " + nombreMiembro + " en el proyecto " + nombreProyecto +

" generado el " + fechaGeneracion + ":\n" + contenido;

}

}