OBLIGATORIO 1

Desarrollo de Aplicaciones (DDA) 2025

Alumno: Alejo De León
Fecha de entrega: 20/10/2025
Materia: Desarrollo de Dispositivos y Aplicaciones
Profesor: Nahuel Pages

Indice

Letra del Problema

Se desea implementar un sistema para gestionar la reserva de canchas en un club deportivo.

El sistema debe:

- Permitir consultar y administrar canchas, socios y turnos disponibles.
- Registrar reservas de canchas, modificarlas o cancelarlas.
- Registrar el pago de los turnos (congelando la tarifa vigente si hay prepago).
- Registrar extras al finalizar la reserva (iluminación, elementos adicionales, extensión de horario, etc.).

Datos a registrar:

Socios:

- idSocio, nombre, apaterno, amaterno (opcional), num_documento, fecha_nacimiento, teléfono, país.

Canchas:

- idCancha, nombre, deporte, cubierta/descubierta, capacidad, estado (disponible, reservada, ocupada), características (iluminación, piso, gradas, etc.).

Reservas:

- idReserva, socio, cancha, fecha, horaInicio, duración, prepago, fecha de reserva, observaciones.
- No se permiten superposiciones de turnos para la misma cancha en la misma fecha y franja horaria.

Tarifas:

- Fecha de entrada en vigencia, aplicable por deporte o tipo de cancha, validas hasta que se defina una nueva tarifa.

Consultas requeridas:

1. Por deporte
2. Por nombre
3. Por fecha
4. Por condición (cubiertas / descubiertas)
5. Filtrar canchas con y sin reserva previa en una fecha dada

Análisis y Descripción de la Solución

Objetivo:

Desarrollar un sistema en **Java** con **consola** como interfaz, utilizando **ArrayList** para almacenar los datos en memoria, que permita:

- Administrar socios y canchas.
- Registrar y gestionar reservas y pagos.
- Validar solapamiento de turnos y aplicar extras.

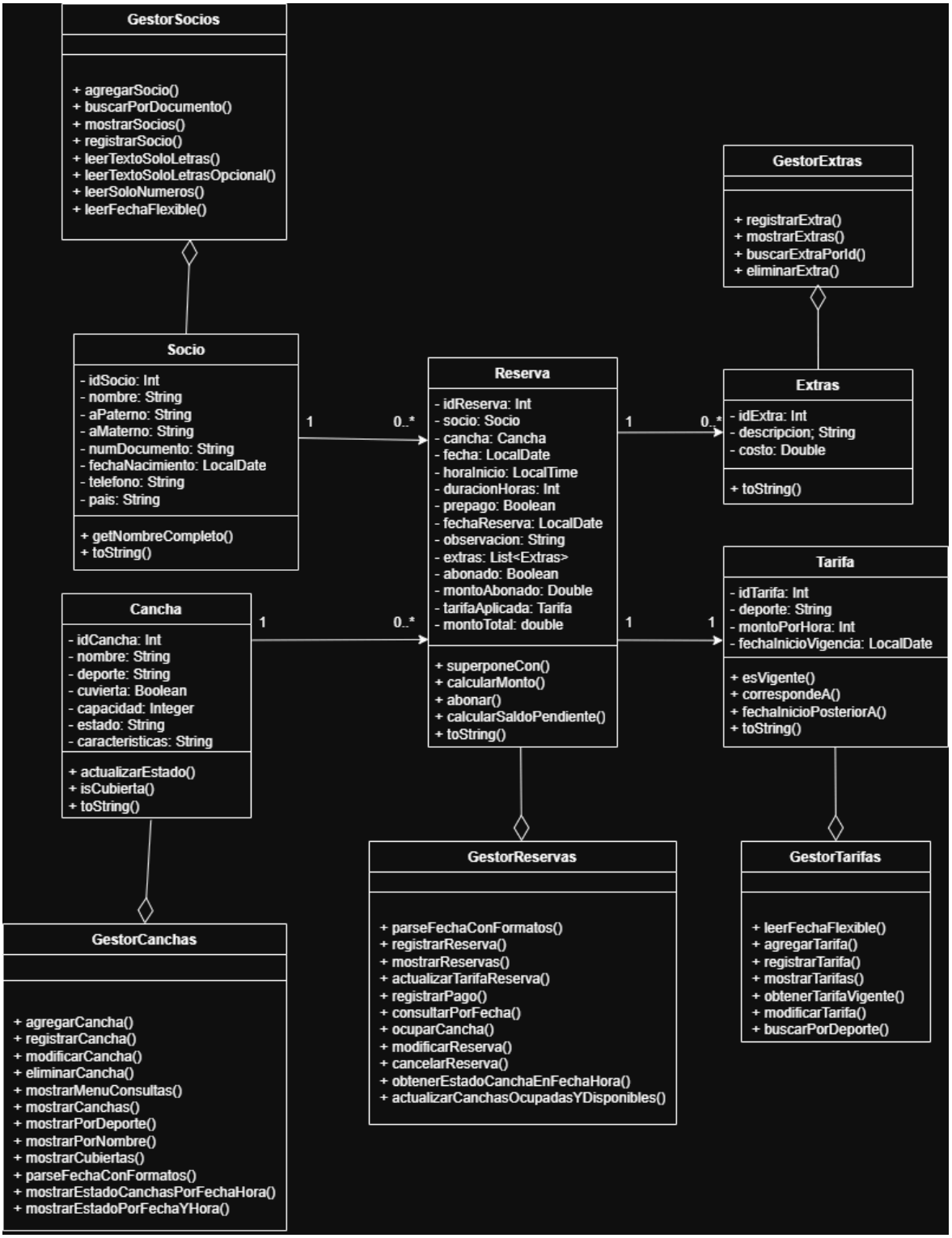
Estructura de Clases propuesta:

- **Socio**: representa a un socio del club.
- **Cancha**: representa la cancha disponible para reserva.
- **Reserva**: representa la reserva de un turno de cancha.
- **Tarifa**: define el precio vigente de un tipo de cancha y deporte.
- **Extra**: opcionales que se cobran al final del turno.
- **GestorSocios**: clase principal para la gestión y lógica de los socios.
- **GestorCanchas**: clase principal para la gestión y lógica de las canchas.
- **GestorTarifas**: clase principal para la gestión y lógica de las tarifas.
- **GestorExtras**: clase principal para la gestión y lógica de las extras.
- **GestorExtras**: clase principal para la gestión y lógica de las reservas.

Reglas de negocio importantes:

- No se permite superposición de reservas.
- Los prepago congelan la tarifa del turno.
- Se deben poder registrar pagos y extras.
- La información de socios y canchas debe ser fácilmente consultable por distintos filtros.

Diagrama de clases



Código de las Clases de Dominio

Clase Socio

```
public class Socio {
    // Atributos de la clase Socio
    private int idSocio;
    private String nombre;
    private String aPaterno;
    private String aMaterno;
    private String numDocumento;
    private LocalDate fechaNacimiento;
    private String telefono;
    private String pais;

    // Constructor de la clase para inicializar todos los atributos
    public Socio(int idSocio, String nombre, String apaterno, String amaterno,
                 String numDocumento, LocalDate fechaNacimiento,
                 String telefono, String pais) {
        this.idSocio = idSocio;
        this.nombre = nombre;
        this.aPaterno = apaterno;
        this.aMaterno = amaterno;
        this.numDocumento = numDocumento;
        this.fechaNacimiento = fechaNacimiento;
        this.telefono = telefono;
        this.pais = pais;
    }

    // Getters y setters para acceder y modificar los atributos
    public int getIdSocio() {
        return idSocio;
    }
    public void setIdSocio(int idSocio) {
        this.idSocio = idSocio;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApaterno() {
```

```
    return aPaterno;
}
public void setApaterno(String apaterno) {
    this.aPaterno = apaterno;
}
public String getAmaterno() {
    return aMaterno;
}
public void setAmaterno(String amaterno) {
    this.aMaterno = amaterno;
}
public String getNumDocumento() {
    return numDocumento;
}
public void setNumDocumento(String numDocumento) {
    this.numDocumento = numDocumento;
}
public LocalDate getFechaNacimiento() {
    return fechaNacimiento;
}
public void setFechaNacimiento(LocalDate fechaNacimiento) {
    this.fechaNacimiento = fechaNacimiento;
}
public String getTelefono() {
    return telefono;
}
public void setTelefono(String telefono) {
    this.telefono = telefono;
}
public String getPais() {
    return pais;
}
public void setPais(String pais) {
    this.pais = pais;
}

// Método para obtener el nombre completo del socio
public String getNombreCompleto() {
    if (aMaterno != null && !aMaterno.isEmpty()) {
        return nombre + " " + aPaterno + " " + aMaterno;
    } else {
        return nombre + " " + aPaterno; // Si no tiene apellido materno
    }
}
```

```
    // Método toString para mostrar los datos del socio como cadena
    @Override
    public String toString() {
        return "Socio {" +
                "IdSocio=" + idSocio +
                ", Nombre='" + nombre + '\'' +
                ", Apellido Paterno='" + aPaterno + '\'' +
                ", Apellido Materno='" + aMaterno + '\'' +
                ", numDocumento='" + numDocumento + '\'' +
                ", fechaNacimiento='" + fechaNacimiento + '\'' +
                ", telefono='" + telefono + '\'' +
                ", pais='" + pais + '\'' +
                '}';
    }
}
```

Clase Cancha

```
public class Cancha {
    // Atributos de la clase Cancha
    private int idCancha;
    private String nombre;
    private String deporte;
    private Boolean cubierta;      // Indica si la cancha es techada
    private Integer capacidad;     // Cantidad de personas que puede albergar
    private String estado;         // Estado actual de la cancha (disponible, en
mantenimiento, etc.)
    private String caracteristicas; // Detalles adicionales de la cancha

    // Constructor para inicializar todos los atributos
    public Cancha(int idCancha, String nombre, String deporte, Boolean cubierta,
                  Integer capacidad, String estado, String caracteristicas) {
        this.idCancha = idCancha;
        this.nombre = nombre;
        this.deporte = deporte;
        this.cubierta = cubierta;
        this.capacidad = capacidad;
        this.estado = estado;
        this.caracteristicas = caracteristicas;
    }

    // Getters y setters para acceder y modificar los atributos
    public int getIdCancha() {
        return idCancha;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getDeporte() {
        return deporte;
    }
    public void setDeporte(String deporte) {
        this.deporte = deporte;
    }

    public Boolean getCubierta() {
```

```
    return cubierta;
}
public void setCubierta(Boolean cubierta) {
    this.cubierta = cubierta;
}
public Integer getCapacidad() {
    return capacidad;
}
public void setCapacidad(Integer capacidad) {
    this.capacidad = capacidad;
}
public String getEstado() {
    return estado;
}
public void setEstado(String estado) {
    this.estado = estado;
}
public String getCaracteristicas() {
    return caracteristicas;
}
public void setCaracteristicas(String caracteristicas) {
    this.caracteristicas = caracteristicas;
}

// Método para actualizar el estado de la cancha
public void actualizarEstado(String nuevoEstado){
    this.estado = nuevoEstado;
}
// Método booleano para saber si la cancha está techada
public boolean isCubierta() {
    return cubierta != null && cubierta;
}
// Método toString para mostrar la información de la cancha como cadena
@Override
public String toString() {
    return "Cancha {" +
            "IdCancha=" + idCancha +
            ", Nombre='" + nombre + '\'' +
            ", Deporte='" + deporte + '\'' +
            ", Cubierta='" + cubierta + '\'' +
            ", Capacidad='" + capacidad + '\'' +
            ", Estado='" + estado + '\'' +
            ", Caracteristicas='" + caracteristicas + '\'' +
            '}';
```

```
}}
```

Clase Tarifa

```
public class Tarifa {
    // Atributos de la clase Tarifa
    private int idTarifa;                // Identificador único de la tarifa
    private String deporte;              // Deporte al que aplica la tarifa
    private double montoPorHora;         // Costo por hora
    private LocalDate fechaInicioVigencia; // Fecha desde la cual la tarifa es válida

    // Constructor para inicializar todos los atributos
    public Tarifa(int idTarifa, String deporte, double montoPorHora, LocalDate
fechaInicioVigencia) {
        this.idTarifa = idTarifa;
        this.deporte = deporte;
        this.montoPorHora = montoPorHora;
        this.fechaInicioVigencia = fechaInicioVigencia;
    }

    // Método que verifica si la tarifa está vigente en una fecha determinada
    public boolean esVigente(LocalDate fecha) {
        return !fecha.isBefore(this.fechaInicioVigencia);
    }

    // Método que verifica si la tarifa corresponde a un deporte específico
    public boolean correspondeA(String deporte) {
        return this.deporte.equalsIgnoreCase(deporte);
    }

    // Getters para acceder a los atributos
    public int getIdTarifa() {
        return idTarifa;
    }
    public String getDeporte() {
        return deporte;
    }
    public double getMontoPorHora() {
        return montoPorHora;
    }
    public LocalDate getFechaInicioVigencia() {
        return fechaInicioVigencia;
    }
```

```
    // Compara la fecha de inicio de esta tarifa con otra tarifa
    public boolean fechaInicioPosteriorA(Tarifa otra) {
        if (otra == null || otra.getFechaInicioVigencia() == null) return true;
        return this.fechaInicioVigencia.isAfter(otra.getFechaInicioVigencia());
    }

    // Método toString para mostrar la tarifa de manera legible
    @Override
    public String toString() {
        return "Tarifa #" + idTarifa + " (" + deporte + "): $" + montoPorHora + "/hora
desde " + fechaInicioVigencia;
    }
}
```

Clase Extras

```
public class Extras {
    // Atributos de la clase Extras
    private int idExtra;          // Identificador único del extra
    private String descripcion;   // Descripción del extra
    private double costo;         // Costo del extra

    // Constructor para inicializar todos los atributos
    public Extras(int idExtra, String descripcion, double costo) {
        this.idExtra = idExtra;
        this.descripcion = descripcion;
        this.costo = costo;
    }

    // Getters para acceder a los atributos
    public int getIdExtra() {
        return idExtra;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public double getCosto() {
        return costo;
    }

    // Método toString para mostrar el extra de manera legible
    @Override
    public String toString() {
        return descripcion + " ($" + costo + ")";
    }
}
```

Clase Reserva

```
public class Reserva {
    // Atributos de la clase Reserva
    private int idReserva;              // Identificador único de la reserva
    private Socio socio;                // Socio que realiza la reserva
    private Cancha cancha;              // Cancha reservada
    private LocalDate fecha;            // Fecha de la reserva
    private LocalTime horaInicio;       // Hora de inicio de la reserva
    private int duracionHoras;          // Duración en horas
    private boolean prepago;            // Indica si se hizo prepago
    private LocalDate fechaReserva;     // Fecha en que se registró la reserva
    private String observacion;         // Observaciones adicionales
    private List<Extras> extras;        // Lista de servicios extra
    private boolean abonado;            // Indica si se ha abonado
    private double montoAbonado;        // Monto abonado por el socio
    private Tarifa tarifaAplicada;      // Tarifa que aplica a la reserva
    private double montoTotal;          // Monto total calculado

    // Constructor para inicializar todos los atributos
    public Reserva(int idReserva, Socio socio, Cancha cancha, LocalDate fecha,
LocalTime horaInicio,
                   int duracionHoras, boolean prepago, LocalDate fechaReserva, String
observacion,
                   List<Extras> extras, Tarifa tarifaAplicada) {

        this.idReserva = idReserva;
        this.socio = socio;
        this.cancha = cancha;
        this.fecha = fecha;
        this.horaInicio = horaInicio;
        this.duracionHoras = duracionHoras;
        this.prepago = prepago;
        this.fechaReserva = fechaReserva;
        this.observacion = observacion;
        this.extras = extras;
        this.tarifaAplicada = tarifaAplicada;
        this.abonado = prepago;                    // Si es prepago, ya se considera
abonado
        this.montoTotal = calcularMonto(this.tarifaAplicada); // Calcula monto total
inicial
    }
```

```
    // Getters y setters para los atributos
    public int getIdReserva() { return idReserva; }
    public Socio getSocio() { return socio; }
    public Cancha getCancha() { return cancha; }
    public LocalDate getFecha() { return fecha; }
    public LocalTime getHoraInicio() { return horaInicio; }
    public int getDuracionHoras() { return duracionHoras; }
    public boolean isPrepago() { return prepago; }
    public LocalDate getFechaReserva() { return fechaReserva; }
    public String getObservacion() { return observacion; }
    public List<Extras> getExtras() { return extras; }
    public boolean isAbonado() { return abonado; }
    public double getMontoAbonado() { return montoAbonado; }
    public Tarifa getTarifaAplicada() { return tarifaAplicada; }

    public void setTarifaAplicada(Tarifa tarifaAplicada) {
        this.tarifaAplicada = tarifaAplicada;
        this.montoTotal = calcularMonto(this.tarifaAplicada); // Recalcula monto si
cambia tarifa
    }

    public double getMontoTotal() {
        double total = (tarifaAplicada != null ? tarifaAplicada.getMontoPorHora() *
duracionHoras : 0);
        for (Extras e : extras) {
            total += e.getCosto();  // Suma el costo de los extras
        }
        return total;
    }

    public void setFecha(LocalDate fecha) {
 this.fecha = fecha;
 }
    public void setHoraInicio(LocalTime horaInicio) {
 this.horaInicio = horaInicio;
 }
    public void setDuracionHoras(int duracionHoras) {
        this.duracionHoras = duracionHoras;
        this.montoTotal = calcularMonto(this.tarifaAplicada); // Actualiza monto si
cambia duración
    }
    public void setAbonado(boolean abonado) {
this.abonado = abonado;
 }
```

```
    public void setMontoAbonado(double montoAbonado) {
 this.montoAbonado = montoAbonado;
 }

    // Verifica si esta reserva se superpone con otra reserva (misma cancha y
fecha)
    public boolean superponeCon(Reserva otra) {
        if (!this.cancha.equals(otra.cancha) || !this.fecha.equals(otra.fecha))
            return false;

        LocalTime fin = horaInicio.plusHours(duracionHoras);
        LocalTime finOtra = otra.horaInicio.plusHours(otra.duracionHoras);

        return !(fin.isBefore(otra.horaInicio) || finOtra.isBefore(this.horaInicio));
    }

    // Calcula el monto total de la reserva incluyendo extras
    public double calcularMonto(Tarifa tarifa) {
        Tarifa usada = (tarifaAplicada != null) ? tarifaAplicada : tarifa;
        double total = (usada != null ? usada.getMontoPorHora() : 0) *
duracionHoras;
        for (Extras e : extras) {
            total += e.getCosto();
        }
        return total;
    }

    // Marca la reserva como abonada
    public void abonar() {
this.abonado = true;
 }

    // Calcula el saldo pendiente de la reserva
    public double calcularSaldoPendiente() {
        return getMontoTotal() - montoAbonado;
    }

    // Representación en texto de la reserva
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Reserva registrada correctamente:\n");
        sb.append("ID Reserva: ").append(idReserva).append("\n");
        sb.append("Socio: ").append(socio.getNombreCompleto()).append("\n");
```

```
        sb.append("Cancha: ").append(cancha.getNombre()).append("
(").append(cancha.getDeporte()).append(")\n");
        sb.append("Fecha: ").append(fecha).append("\n");
        sb.append("Hora inicio: ").append(horaInicio).append("\n");
        sb.append("Duración: ").append(duracionHoras).append("h\n");
        sb.append("Prepago: ").append(prepago ? "Sí" : "No").append("\n");

        if (!extras.isEmpty()) {
            sb.append("Extras:\n");
            for (Extras e : extras) {
                sb.append(" - ").append(e.getDescripcion()).append("
($").append(e.getCosto()).append(")\n");
            }
        }

        sb.append("Monto total: $").append(getMontoTotal()).append("\n");
        sb.append("Monto abonado: $").append(montoAbonado).append("\n");
        sb.append("Saldo pendiente:
$").append(calcularSaldoPendiente()).append("\n");
        sb.append("Observaciones: ").append(observacion.isEmpty() ? "ninguna" :
observacion);

        return sb.toString();
    }
}
```

Consultas

Consulta de canchas por deporte

```
public List<Cancha> buscarPorDeporte(String deporte) {
    List<Cancha> resultado = new ArrayList<>();
    for (Cancha c : canchas) {
        if (c.getDeporte().equalsIgnoreCase(deporte)) {
            resultado.add(c);
        }
    }
    return resultado;
}
```

Consulta de canchas por nombre

```
public List<Cancha> buscarPorNombre(String nombre) {
    List<Cancha> resultado = new ArrayList<>();
    for (Cancha c : canchas) {
        if (c.getNombre().equalsIgnoreCase(nombre)) {
            resultado.add(c);
        }
    }
    return resultado;
}
```

Consulta de canchas por condición (cubiertas o descubiertas)

```
public List<Cancha> buscarPorCondicion(boolean cubierta) {
    List<Cancha> resultado = new ArrayList<>();
    for (Cancha c : canchas) {
        if (c.isCubierta() == cubierta) {
            resultado.add(c);
        }
    }
    return resultado;
}
```

Consulta de canchas disponibles en una fecha (por reservas)

```
   public List<Cancha> buscarPorFecha(LocalDate fecha, List<Reserva>
reservas) {
       List<Cancha> disponibles = new ArrayList<>(canchas);
       for (Reserva r : reservas) {
          if (r.getFecha().equals(fecha)) {
             disponibles.remove(r.getCancha());
          }
       }
       return disponibles;
   }
}
```

Registrar una nueva reserva (validando superposición)

```
   public boolean registrarReserva(Reserva nuevaReserva) {
       for (Reserva r : reservas) {
          if (r.getCancha().equals(nuevaReserva.getCancha()) &&
             r.getFecha().equals(nuevaReserva.getFecha())) {

             LocalTime iniExistente = r.getHoraInicio();
             LocalTime finExistente = iniExistente.plusHours(r.getDuracion());
             LocalTime iniNueva = nuevaReserva.getHoraInicio();
             LocalTime finNueva =
iniNueva.plusHours(nuevaReserva.getDuracion());

             boolean seSuperpone = !(finNueva.isBefore(iniExistente) ||
iniNueva.isAfter(finExistente));
             if (seSuperpone) {
                return false; // superposición detectada
             }
          }
       }
       reservas.add(nuevaReserva);
       return true;
   }
```

Filtrar canchas con reserva previa en una fecha dada

```
    public List<Cancha> canchasConReserva(LocalDate fecha) {
        List<Cancha> conReserva = new ArrayList<>();
        for (Reserva r : reservas) {
            if (r.getFecha().equals(fecha)) {
                conReserva.add(r.getCancha());
            }
        }
        return conReserva;
    }
```

Filtrar canchas sin reserva previa en una fecha dada

```
    public List<Cancha> canchasSinReserva(LocalDate fecha, List<Cancha>
todasCanchas) {
        List<Cancha> conReserva = canchasConReserva(fecha);
        List<Cancha> sinReserva = new ArrayList<>(todasCanchas);
        sinReserva.removeAll(conReserva);
        return sinReserva;
    }
}
```

Repositorio del Proyecto en GitHub

Durante el desarrollo del sistema, se utilizó GitHub como herramienta principal de control de versiones y colaboración.

En este repositorio se encuentra todo el código fuente del proyecto, junto con los diagramas UML, documentación complementaria y versiones anteriores del desarrollo.

Repositorio: https://github.com/AlejoDL017/ObligatorioDDA---Alejo-De-Leon

Estructura general:

- src/: Código fuente en Java.
- documentacion/: Entrega final en formato PDF.
- diagramas/: Archivo .uml
- README.md: Descripción general del sistema e instrucciones de ejecución.

Este repositorio permite acceder fácilmente al proyecto completo, verificar el historial de commits y comprobar la evolución del desarrollo.