Ingeniería en Informática

Instituto de Ingeniería y Agronomía

Metodologías de programación 2

Trabajo final

Informe

DOCENTE: Lic. Claudia Cappelletti

Integrantes: Lapegna Iván
Oviedo Federico
Salas Alejandro
Arnés Laura

COMISIÓN: 1

Índice

Introducción

En este informe se presenta la aplicación de distintos conceptos y metodologías trabajadas a lo largo de la cursada de la materia *Metodologías de la Programación II*. El enfoque principal está centrado en el desarrollo de una aplicación utilizando el lenguaje Smalltalk, específicamente con el entorno Dolphin Smalltalk.

Comenzaremos con una breve descripción del contexto que dio origen a la propuesta de desarrollo. Luego, se detallará la especificación e implementación de cada una de las clases que conforman el sistema, así como su correspondiente código. Finalmente, se expondrán las metodologías de trabajo adoptadas por el equipo.

Descripción

El objetivo principal del sistema desarrollado consiste en optimizar el proceso de gestión de reservas y canchas de fútbol, integrando diversos conceptos aprendidos durante la cursada de la materia.
El sistema surge a partir de la necesidad de gestionar de forma eficiente las reservas de canchas en un complejo deportivo. Cuando una persona desee reservar una cancha, el encargado debe acceder al sistema y completar una serie de datos, como el número del cliente, fecha del día a reservar, cancha deseada y horario. A medida que el usuario vaya ingresando los datos se realizan distintas validaciones, como si hay alguna cancha disponible el día elegido y se mostrarán solo los horarios disponibles. Si el mensaje se cumple exitosamente se genera la reserva a nombre del cliente, se marca el horario en la cancha elegida como ocupado y se informa que la reserva fue realizada con éxito.

Diagrama UML

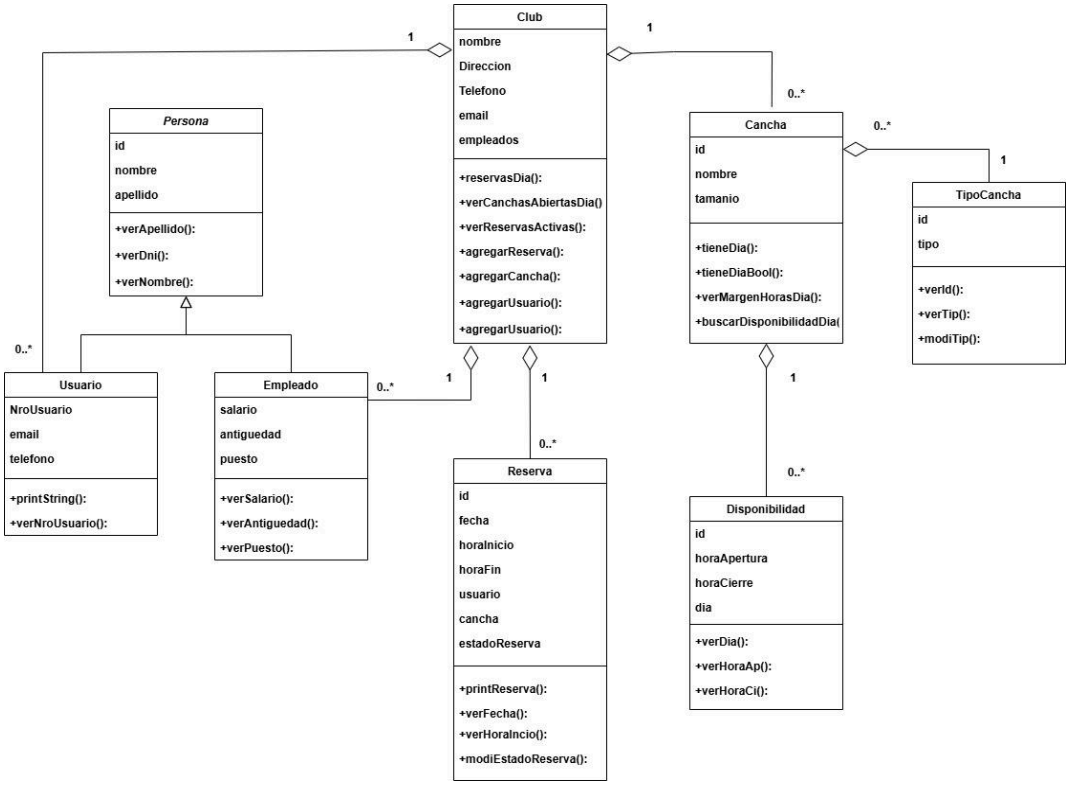
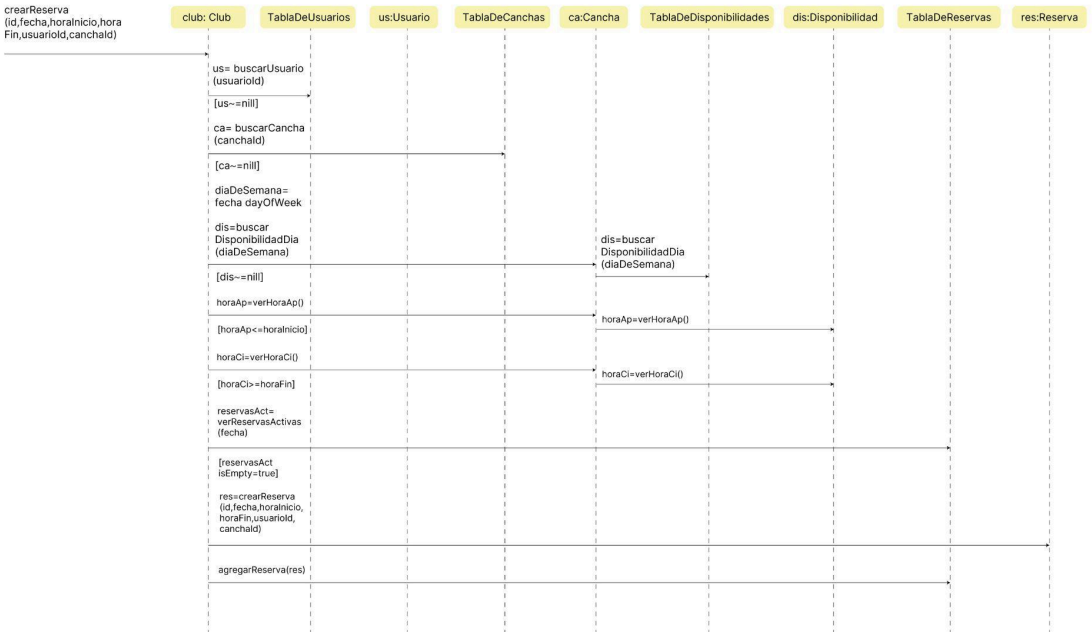


Diagrama de secuencias



Especificación e implementación de las clases utilizadas

Especificación

Club	
Atributos	Métodos
Nombre	agregarCancha
Dirección	agregarEmpleado
Teléfono	agregarReserva
Email	agregarUsuario
Empleados	buscarCancha
Usuarios	buscarEmpleado
Reservas	buscarReserva
Canchas	buscarUsuario
	cantidadCanchas
	cantidadEmpleados
	cantidadReservas
	cantidadUsuarios
	initClub
	modiDir
	modiEmail
	modiNom
	modiTel
	reservasDia
	verDir
	verEmail
	verNom
	verReservasActivas

	verTel
	verTodasLasCanchas
	verTodasLasReservas
	verTodosLosEmpleados
	verTodosLosUsuarios

Cancha	
Atributos	Métodos de instancia
Id	crearCancha
Nombre	agregarDisponobilidad
Tamaño	buscarDisponibilidad
Tipo	buscarDisponibilidadDia
Disponibilidades	cantidadDisponibilidades
	initCancha
	modiNomb
	modiTam
	modiTipo
	tieneDia
	tieneDiaBool
	verId
	verMargenHorasDia
	verNomb
	verTam
	verTip
	verTodaslasDisponibilidades

TipoCancha	
Atributos	Métodos de instancia
Id	crearTipoCancha
Tipo	initTipo
	modiTip
	verId
	verTip

Disponibilidad	
Atributos	Métodos de instancia
Id	crearDisponibilidad
HoraApertura	initdisponibilidad
HoraCierre	modiDia
Dia	modiHoraAp
	modiHoraCi
	verDia
	verHoraAp
	verHoraCi
	verId

Reserva	
Atributos	Métodos de instancia
Id	crearReserva
Fecha	initReserva
HoraInicio	modiCancha
HoraFin	modiEstadoReserva

Usuario	modiFecha
Cancha	modiHoraFina
EstadoReserva	modiHoraInicio
	modiId
	modiUsuario
	printReserva
	verCancha
	verEstadoreserva
	verFecha
	verHoraFin
	verHoraInicio
	verid
	verUsuario

Persona	
Atributos	Métodos de instancia
Dni	crearPersona
Nombre	initPersona
Apellido	verApellido
	verDni
	verNombre

Usuario	
Atributos	Métodos de instancia
NroUsuario	crearUsuario
Email	initUsuario
Telefono	modificarEmail
	modificarTelefono

	printString
	verEmail
	verNroUsuario
	verTelefono

Empleado	
Atributos	Métodos de instancia
Salario	crearEmpleado
Antiguedad	initEmpleado
Puesto	modificarPuesto
	modificarSalario
	verAntiguedad
	verPuesto
	verSalario

Implementación

Club

Método de clase

Método que permite instanciar un objeto de la clase Club

```
crearClubnom: unNom dir: unaDir tel: unTel email: unEmail
^(self new) initClubnom: unNom dir: unaDir tel: unTel email: unEmail.
```

Métodos de Instancia

Método para inicializar un club, se asignan los atributos pasados por parámetro, y se crean las colecciones.

```
initClubnom:unNom dir:unaDir tel:unTel email:unEmail
nombre:= unNom.
direccion:= unaDir.
telefono:= unTel.
email:= unEmail.
empleados:= OrderedCollection new.
usuarios:= OrderedCollection new.
reservas := OrderedCollection new.
canchas:= OrderedCollection new.
```

Métodos que permiten visualizar los datos de la clase.

```
verDir
^direccion.
```

```
verEmail
^email.
```

```
verNom
^nombre.
```

```
verTel
^telefono.
```

Método que permite visualizar las colecciones de la clase.

```
verTodasLasCanchas
^canchas.
```

```
verTodasLasReservas
^reservas.
```

```
verTodosLosEmpleados
^empleados.
```

```
verTodosLosUsuarios
^usuarios.
```

Métodos que permiten modificar los datos de la clase.

```
modiDir:unaDir
direccion:=unaDir.
```

```
modiEmail:unEmail
email:=unEmail.
```

```
modiNom:unNom
nombre:=unNom.
```

```
modiTel:unTel
telefono:=unTel.
```

Métodos que permiten agregar elementos a las colecciones de la clase.

```
agregarCancha:unCancha
canchas add: unCancha.
```

```
agregarEmpleado:unEmpleado
empleados add: unEmpleado.
```

```
agregarReserva:unReserva
reservas add: unReserva.
```

```
agregarUsuario:unUsuario
usuarios add: unUsuario.
```

Métodos que permiten buscar elementos en las colecciones de la clase.

```
buscarCancha:idCancha
^ canchas detect: [:cancha | cancha verId = idCancha ] ifNone: [nil].
```

```
buscarEmpleado:idEmpleado
^empleados detect: [:empleado | empleado verId = idEmpleado ] ifNone: [nil].
```

```
buscarReserva:idReserva
^reservas detect: [:reserva | reserva verId = idReserva ] ifNone: [nil].
```

```
buscarUsuario:idUsuario
^usuarios detect: [:usuario | usuario verNroUsuario = idUsuario ] ifNone: [nil].
```

Métodos que permiten saber cuántos elementos tienen las colecciones de la clase.

```
cantidadCanchas
^canchas size.
```

```
cantidadEmpleados
^empleados size.
```

```
cantidadReservas
^reservas size.
```

```
cantidadUsuarios
^usuarios size.
```

Método que permite encontrar y devolver todas las reservas de un club que corresponden a una fecha específica.

```
reservasDia: unaFecha
|reservas|
reservas := self verTodasLasReservas.
^reservas select: [:reserva | reserva verFecha = unaFecha ].
```

Cancha

Método de clases

Método que permite instanciar un objeto de la clase Cancha.

```
crearCanchaid:unId nom:unNom tam:unTam tip:unTip
^(self new) initCanchaid:unId nom:unNom tam:unTam tip:unTip.
```

Método de instancia

Método para inicializar una cancha, se asignan los atributos pasados por parámetro, y se crean las colecciones.

```
initCanchaid:unId nom:unNom tam:unTam tip:unTip
id:= unId.
nombre := unNom.
tamanio := unTam.
tipo := unTip .
disponibilidades := OrderedCollection new.
```

Métodos que permiten visualizar los datos de la clase.

```
verId
^id.
```

```
verNomb
^nombre.
```

```
verTam
^tamanio.
```

```
verTip
^tipo.
```

Método que permite visualizar la colección de la clase.

```
verTodasLasDisponibilidades
^disponibilidades.
```

Métodos que permiten modificar los datos de la clase.

```
modiNomb:unNomb
nombre :=unNomb.
```

```
modiTam:unTam
tamanio :=unTam.
```

```
modiTipo:unTipo
tipo :=unTipo.
```

Método que permite agregar disponibilidades a la colección de la clase.

```
agregarDisponibilidad:unaDisp
disponibilidades add: unaDisp.
```

Métodos que permiten buscar disponibilidades por id y por día en la colección de la clase.

```
buscarDisponibilidad:idDisponibilidad
^ disponibilidades detect: [:disponibilidad | disponibilidad verId = idDisponibilidad ] ifNone: [nil].
```

```
buscarDisponibilidadDia:unDia
^ disponibilidades detect: [:disponibilidad | disponibilidad verDia = unDia ] ifNone: [nil].
```

Método que permite saber cuántos elementos tiene la colección de la clase.

```
cantidadDisponibilidades
^disponibilidades size.
```

Método que permite buscar disponibilidades por día.

```
tieneDia:unDia
^self verTodasLasDisponibilidades detect: [:disp | disp verDia = unDia] ifNone: [nil].
```

Método que revisa la colección de la clase para ver si al menos un elemento cumple la condición de que su día seas igual al pasado por parámetro.

```
tieneDiaBool: unDia
    ^ (self verTodasLasDisponibilidades anySatisfy: [:disp | disp verDia = unDia]).
```

Método que calcula el rango de disponibilidad horaria de un día específico.

```
verMargenHorasDia: unDia
^((self tieneDia: unDia) verHoraCi)  - ((self tieneDia: unDia) verHoraAp).
```

TipoCancha

Método de clases

Método que permite instanciar un objeto de la clase TipoCancha.

```
crearTipoCanchaid: unId tip:unTip
^(self new) initTipoid:unId tip:unTip.
```

Método de instancia

Método para inicializar un tipo de cancha, se asignan los atributos pasados por parámetro.

```
initTipoid:unid tip:unTip
id := unid.
tipo := unTip.
```

Métodos que permiten visualizar los datos de la clase.

```
verId
^id.
```

```
verTip
^tipo.
```

Métodos que permiten modificar los datos de la clase.

```
modiTipo:unTipo
tipo :=unTipo.
```

Reserva

Método de clase

Método que permite instanciar un objeto de la clase Reserva.

```
crearReservaId:unId fecha:unaFecha horaInicio:unaHoraInicio horaFin:unaHoraFin usuario:unUsuario cancha:unaCancha
^(self new) initReservaId: unId fecha: unaFecha horaInicio: unaHoraInicio horaFin: unaHoraFin usuario: unUsuario cancha: unaCancha.
```

Método de instancia

Método para inicializar una cancha, se asignan los atributos pasados por parámetro, y se pone el estado de la reserva en true.

```
initReservaId:unId fecha:unaFecha horaInicio:unaHoraInicio horaFin:unaHoraFin usuario:unUsuario cancha:unaCancha
id:=unId.
fecha:=unaFecha.
horaInicio:=unaHoraInicio.
horaFin:=unaHoraFin.
usuario:=unUsuario.
cancha:=unaCancha.
estadoReserva:=true.
```

Métodos que permiten visualizar los datos de la clase.

```
verCancha
^cancha.
```

```
verEstadoReserva
^estadoReserva.
```

```
verFecha
^fecha.
```

```
verHoraFin
^horaFin.
```

```
verHoraInicio
^horaInicio.
```

```
verId
^id.
```

```
verUsuario
^usuario.
```

Métodos que permiten modificar los datos de la clase.

```
modiCancha:unaCancha
cancha:=unaCancha.
```

```
modiEstadoReserva:unEstadoReserva
estadoReserva:=unEstadoReserva.
```

```
modiFecha:unaFecha
fecha:=unaFecha.
```

```
modihoraFin:unaHoraFin
horaFin:=unaHoraFin.
```

```
modihoraInicio:unaHoraInicio
horaInicio:=unaHoraInicio.
```

```
modiId:unId
id:=unId.
```

```
modiUsuario:unUsuario
usuario:=unUsuario.
```

Método que imprime los detalles de una reserva

```
printReserva: unClub
      |u c|
      u := unClub buscarUsuario: usuario.
      c := (unClub buscarCancha: cancha).
   ^ 'Horario: ', horaInicio printString , ' a ', horaFin printString , Character cr asString,
     u  printString, Character cr asString,
     'Cancha de Fútbol ',(c verTam) printString, ' - ', (c verTip) verTip ,Character cr asString,
     '------------------------------------------------------------ '
```

Disponibilidad

Método de clase

Método que permite instanciar un objeto de la clase Disponibilidad

```
crearDisponibilidadid:unId horaAp:unaHoraAp horaCi:unaHoraCi dia:unDia
^(self new) initDisponibilidadid:unId horaAp:unaHoraAp horaCi:unaHoraCi dia:unDia.
```

Métodos de Instancia

Método para inicializar una disponibilidad, se asignan los atributos pasados por parámetro.

```
initDisponibilidadid:unId horaAp:unaHoraAp horaCi:unaHoraCi dia:unDia
id:= unId.
horaApertura := unaHoraAp.
horaCierre := unaHoraCi.
dia := unDia.
```

Métodos que permiten visualizar los datos de la clase.

```
verDia
^dia.
```

```
verHoraAp
^horaApertura.
```

```
verHoraCi
^horaCierre.
```

```
verId
^id.
```

Métodos que permiten modificar los datos de la clase.

```
modiDia:unDia
dia :=unDia.
```

```
modiHoraAp:unaHoraAp
horaApertura :=unaHoraAp.
```

```
modiHoraCi:unaHoraCi
horaCierre :=unaHoraCi.
```

Persona

Método de clase

Método que permite instanciar un objeto de la clase persona

```
crearPersonadni:unDni nom:unNom apellido:unApe
^(self new) initPersonadni: unDni nom: unNom apellido: unApe.
```

Métodos de instancia

Método para inicializar una persona, se asignan los atributos pasados por parámetro.

```
initPersonadni:unDni nom:unNom apellido:unApe
dni:=unDni.
nombre:=unNom.
apellido:=unApe.
```

Métodos que permiten visualizar los datos de la clase.

```
verApellido
^apellido.
```

```
verDni
^dni.
```

```
verNombre
^nombre.
```

Usuario

Método de clase

Método que permite instanciar un objeto de la clase Usuario.

```
crearUsuariodni: unDni nombre: unNom apellido: unApe NroUsuario: unUsu Email: unEma
Telefono: unTel
^(self new) initUsuariodni: unDni nombre: unNom apellido: unApe nroUsuario: unUsu email:
unEma telefono: unTel.
```

Métodos de instancia

Método para inicializar un empleado, se asignan los atributos pasados por parámetro.

```
initUsuariodni: unDni nombre: unNom apellido: unApe nroUsuario: unUsu email: unEma telefono: unTel
   super initPersonadni: unDni nom: unNom apellido: unApe.
   NroUsuario := unUsu.
   Email := unEma.
   Telefono := unTel.
```

Métodos que permiten visualizar los datos de la clase.

```
verEmail
^Email.
```

```
verNroUsuario
^NroUsuario.
```

```
verTelefono
^Telefono.
```

Método que imprime el nombre completo de un usuario

```
printString
   ^ 'Usuario: ',nombre, ' ', apellido
```

Métodos que permiten modificar los datos de la clase.

```
ModificarEmail: unema
Telefono := unema.
```

```
ModificarTelefono: untel
Telefono := untel.
```

Empleado

Método de Clase

Método que permite instanciar un objeto de la clase Empleado.

```
crearEmpleadoDni: unDni Nombre: unNom Apellido: unApe Salario: unSal Antiguedad: unAnt puesto: unPue
  ^(self new) initEmpleadodni: unDni nombre: unNom apellido: unApe salario: unSal antiguedad: unAnt puesto: unPue
```

Métodos de instancia:

Método para inicializar un empleado, se asignan los atributos pasados por parámetro.

```
initEmpleadodni: unDni nombre: unNom apellido: unApe salario: unSal antiguedad: unAnt puesto: unPue
  super initPersonadni: unDni nom: unNom apellido: unApe.
  salario := unSal.
  antiguedad := unAnt.
  puesto := unPue.
```

Métodos que permiten visualizar los datos de la clase.

```
verAntiguedad
^antiguedad.
```

```
verpuesto
^puesto.
```

```
verSalario
^salario.
```

Métodos que permiten modificar los datos de la clase.

```
modificarPuesto: unPue
puesto := unPue.
```

```
modificarSalario: unSal
salario := unSal.
```

Implementación de la aplicación

```
Transcript clear;
show: '--------------------------------------------------------------------------------' ;
cr;
show: '------------ Sistema de reservas de canchas de  ',  club verNom , ' ------------';
cr;
show: '--------------------------------------------------------------------------------';
cr.
Transcript show: 'Opciones: '; cr;
show: '1 - Crear reserva.'; cr;
show: '2 - Crear cancha.'; cr;
show: '3 - Crear usuario.'; cr;
show: '4 - Crear Empleado.'; cr;
show: '5 - Cancelar reserva.'; cr;
show: '6 - Agregar disponibilidad a una cancha.'; cr;
show: '7 - Mostrar usuarios.'; cr;
show: '8 - Mostrar reservas por día.'; cr;
show: '9 - Mostrar cantidad de reservas por usuario.'; cr;
show: '10 - Mostrar cantidad de reservas por cancha.'; cr;
show: '0 - Salir del programa.'; cr;
show: '--------------------------------------------------------------------------------------------------------'; cr.

opc:= (Prompter prompt: ' Ingrese una opcion:') asNumber.
[opc ~= 0] whileTrue: [
```

El menú implementado en el sistema cuenta con diez opciones principales. Las primeras cuatro están destinadas a la creación de instancias de distintos objetos dentro de la aplicación. La quinta opción permite cancelar una reserva, lo cual resulta útil en caso de que surja algún imprevisto que impida al usuario hacer uso de la misma.

Por otro lado, la opción número seis permite agregar un horario adicional a una cancha, lo que resulta especialmente práctico si una reserva necesita extenderse más allá del tiempo originalmente asignado. Finalmente, las opciones restantes están orientadas a la visualización de información, como la cantidad total de usuarios o las reservas registradas en un día específico. Estas funcionalidades son valiosas para llevar un control organizado tanto de los usuarios como de las reservas realizadas.

La estructura del menú está construida a partir de un bucle `while`, que se mantiene activo hasta que el usuario ingresa el valor 0 para finalizar la ejecución. Además, se emplea una estructura condicional `if` para gestionar el comportamiento correspondiente a cada una de las diez opciones disponibles.

1 - Crear reserva

Esta sección del menú permite al usuario registrar una nueva reserva. Para ello, primero se solicita la fecha deseada, validando que no sea anterior a la fecha actual. Una vez ingresada correctamente, se determina el día de la semana para verificar qué canchas están disponibles en ese día.

El sistema contempla dos posibles escenarios:

Caso 1, No existen reservas para la fecha seleccionada: Se muestran todas las canchas que están habilitadas ese día de la semana. El usuario puede seleccionar una de ellas y luego debe elegir un horario dentro de la franja horaria disponible.

Caso 2, Ya existen reservas en esa fecha: En este caso, el sistema analiza cada cancha para determinar si aún cuenta con horarios disponibles. Para ello, se arma un diccionario que relaciona cada cancha con sus reservas en la fecha elegida. Si una cancha ya tiene todos sus turnos ocupados, se descarta. También se identifican las canchas que están habilitadas ese día pero aún no tienen reservas.

Una vez seleccionada la cancha, la fecha y el horario, el usuario debe ingresar su número de usuario. El sistema verifica que el número exista en el registro. Si es válido, se crea una nueva instancia de reserva

2 - Crear cancha

Esta opción es más sencilla que la anterior, consta de 3 pasos simples, primero se debe introducir el nombre de la cancha, después se debe seleccionar el tamaño de la cancha variando de entre canchas de futbol 5, 7 o 11 para finalmente agregar el tipo de cancha que va a ser, si va a ser de césped, sintética o una cancha de sala.

3 - Crear usuario

En esta sección se puede crear a los usuarios, para crear uno se debe ingresar la información solicitada como el nombre completo, el dni y el numero de telefono con el mail para poder contactarlo, cuando se ingresa todos los datos, al usuario se le asigna un número de usuario con su orden de llegada y luego se pregunta si se quiere crear un nuevo usuario, si la elección es afirmativa se puede crear otro pero si es negativa se termina la creación de usuarios.

4 - Crear empleado

Esta función es similar a la anterior ya que también requiere de crear a una subclase de la superclase persona, asi que tambien requiere de que se le ingrese el nombre completo de la persona además del dni y como dato distinto se le pide que se ingrese la antigüedad si la persona tiene, luego se da a elegir entre los puestos que puede ocupar la persona, finalmente cuando se creó la instancia, se pregunta si quiere agregar otro empleado preguntando si quiere o no quiere.

5 - Cancelar reserva

Lo que hace esta opción es simple, primero pide el id de la reserva que se quiere cancelar, luego se busca la reserva que tiene ese id y se elimina quedando el horario libre para otra posible reserva.

6 - Agregar disponibilidad a una cancha

Esta opción permite asignar un nuevo horario de apertura y cierre a una cancha en un día específico de la semana. Primero, el usuario elige la cancha y el día, el sistema averigua que no exista una disponibilidad para ese día. Luego, se ingresan las horas de apertura y cierre, las cuales deben respetar un orden lógico, finalmente se pregunta si se quiere agregar otra disponibilidad.

7 - Mostrar usuarios

En esta opción se muestran todos los usuarios de manera ordenada gracias a el uso de una colección de tipo sortedCollection. Primero, pide a los usuarios de la clase club y los ingresa en la colección para que se guarden ordenados por su apellido y luego con un do va recorriendo la lista e imprimiendo los datos de cada usuario.

8 - Mostrar reservas por dia

con esta opción se puede visualizar la cantidad de reservas que hay en un dia ademas de ver que cancha y horario fueron seleccionados, además de mostrar el nombre del usuario para confirmar que sea el mismo. primero se piden todas las reservas de dicho día al club y luego con un do se recorre la lista y se imprime los datos de la reserva.

9 - Mostrar la cantidad de reservas por usuario

Esta sección permite ver la cantidad de reservas que realizó un usuario hasta el momento, esto lo logra con dos colecciones, una de reservar y otra de usuarios, la de reservas la llena directamente mientras que la de usuarios se llena con los usuarios que hayan registrado reservas.
Luego elimina los posibles usuarios repetidos de la lista y finalmente selecciona de las reservas a aquellas que estén relacionadas a un usuario e imprime la cantidad junto al nombre del mismo.

10 - Mostrar cantidad de reservas por cancha

finalmente, la última sección utiliza un sistema similar a la opción 9, permite ver la cantidad de reservas que hay por cancha y lo logra al utilizar dos listas, una con las canchas y otra con las reservas, luego con un do recorre las canchas y dentro del mismo agrega un do anidado que utiliza para contar la cantidad de reservas que tiene esa cancha, para terminar se imprime el id de la cancha con su nombre y la cantidad de reservas que tiene.

Metodología de trabajo

Para el desarrollo del sistema de gestión de canchas y reservas de fútbol, adoptamos un enfoque de trabajo inspirado en la metodología Kanban, incorporando varias de sus características principales para organizar y visualizar el progreso del proyecto de forma clara y colaborativa.
A lo largo del proyecto, nos apoyamos en un tablero visual, utilizando el software de gestión de proyectos '*ClickUp*' (https://sharing.clickup.com/90131305077/b/h/6-901311844853-2/15253f2c9baef16) dividido en las etapas "Pendiente", "En progreso", "Base Creada" y "Completo". El tablero nos permitió tener un panorama general del estado de cada tarea, ayudándonos a distribuir el trabajo entre los integrantes, siguiendo un orden de tareas.
No se asignaron roles fijos, sino que cada integrante asumio tareas de acuerdo con sus habilidades e interés. La comunicación fue un pilar fundamental, para resolver dudas y coordinar tareas.

Frameworks para el desarrollo orientado a objetos

Se presentan algunos frameworks destacados para el desarrollo orientado a objetos, que facilitan la implementación de aplicaciones, evitando comenzar desde cero y fomentando buenas prácticas de diseño.

- .NET (C#Permite estructurar el código de forma clara y reutilizable, integrando herramientas como ASP.NET para el desarrollo web, Xamarin para aplicaciones móviles y Windows Forms para entornos de escritorio. Su enfoque multiplataforma y su integración con Visual Studio lo convierten en una opción muy compleja para desarrollos orientados a objetos.

- Spring Boot (Java): Framework que simplifica el desarrollo de aplicaciones empresariales en java. Facilita la creación de aplicaciones web y microservicios utilizando una arquitectura orientada a objetos. Proporciona soporte integrado para inyección de dependencias, gestión de bases de datos, entre otras herramientas, mientras promueve la separación de responsabilidades y la reutilización de código.

Casos de uso de Patrones

Patrones de diseño que fueron usados o podrían ser usados en este proyecto:

- **Facade (Fachada):** Este patrón provee una interfaz simplificada para un sistema más complejo. En el proyecto, la clase Club actúa como una fachada, ya que centraliza y simplifica todas las operaciones (agregar usuarios, buscar canchas, crear reservas), ocultando la complejidad de la gestión interna de las colecciones.

- **Iterator (Iterador):** Permite recorrer una colección sin exponer su estructura interna. Este patrón se utilizó masivamente en todo el código a través de los métodos de colección de Smalltalk como do:, select:, reject: y collect: para listar, filtrar y transformar los datos.

- **Singleton (Instancia Única):** Asegura que una clase tenga una sola instancia y provee un punto de acceso global a ella. El objeto club se utiliza como un Singleton, ya que se crea una única vez al inicio y se usa como el punto de acceso central para todas las operaciones del sistema.

- **Factory Method (Método de Fábrica):** Define una interfaz para crear un objeto, pero delega la decisión de qué clase específica crear a las subclases. Se podría haber usado para manejar la creación de distintos tipos de personas sin que el menú principal necesite conocer cada clase concreta.

- **Command (Comando):** Encapsula una solicitud como un objeto, lo que permite registrarla y deshacerla. Se podría haber aplicado en la "cancelación de reservas" para implementar fácilmente una función de "deshacer" la cancelación.

Repositorios para el desarrollo de software colaborativo

Se utilizó Github durante el desarrollo del proyecto como repositorio para el control de versiones y el almacenamiento del código fuente. Permitiéndonos llevar un seguimiento claro y ordenado de los cambios efectuados, facilitando tanto la gestión de versiones como la colaboración entre los integrantes, siendo fundamental para mantener la integridad del código y coordinar el trabajo de forma efectiva a lo largo de todo el proceso de desarrollo de la aplicación.
Se adjunta enlace al repositorio mencionado:
https://github.com/Alejandro-Ariel-Salas/Alquiler-de-canchas