



UNIVERSIDAD DE LAS FUERZAS ARMADAS (ESPE)

SEGUNDO SEMESTRE

CARRERAS TECNICAS

DEPARTAMENTO DE CIENCIAS EXACTAS

PRIMER PARCIAL

“Sistema de Gestión de Proyectos de TIC”

AUTORES:

Steven Guerrero

Samir Samande

NRC:

1322

DOCENTE:

LUIS ENRIQUE JARAMILLO MONTAÑO

SANGOLQUI – ECUADOR

1. Introducción

En el contexto actual, la gestión eficiente de proyectos de Tecnologías de la Información y Comunicación (TIC) es esencial para garantizar el éxito de las iniciativas en un entorno empresarial dinámico y altamente competitivo. La implementación de herramientas automatizadas que permitan organizar, supervisar y actualizar los elementos clave de un proyecto resulta crucial para optimizar los recursos disponibles y mejorar la productividad del equipo.

En este documento, se desarrolla un sistema de gestión de proyectos basado en los principios de Programación Orientada a Objetos (POO). Este sistema está diseñado para registrar proyectos, asignar tareas y monitorear su estado, proporcionando a los usuarios una solución práctica y estructurada para gestionar sus proyectos. A través del uso de clases, atributos y métodos, se busca demostrar la aplicabilidad de los conceptos de POO en un entorno real y funcional.

El desarrollo de este sistema incluirá el diseño de un diagrama UML, la codificación del programa, y un informe que documente el proceso y los resultados obtenidos, cumpliendo con los criterios establecidos para la actividad experimental.

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Diseñar e implementar un sistema de gestión de proyectos de TIC que permita registrar proyectos, asignar tareas a los integrantes del equipo y monitorear el estado de las tareas, utilizando principios de Programación Orientada a Objetos .

3. OBJETIVOS ESPECÍFICO

Desarrollar un sistema de gestión de proyectos de TIC utilizando Programación Orientada a Objetos (POO), que permita registrar proyectos y almacenar información relevante, como nombre, descripción y fecha de inicio.

Implementar una funcionalidad que asigne tareas a los miembros del equipo de cada proyecto, permitiendo establecer responsables, plazos y prioridades para cada tarea.

Diseñar una interfaz que facilite la consulta y actualización del estado de las tareas asignadas, permitiendo a los usuarios monitorear el progreso de los proyectos en tiempo real.

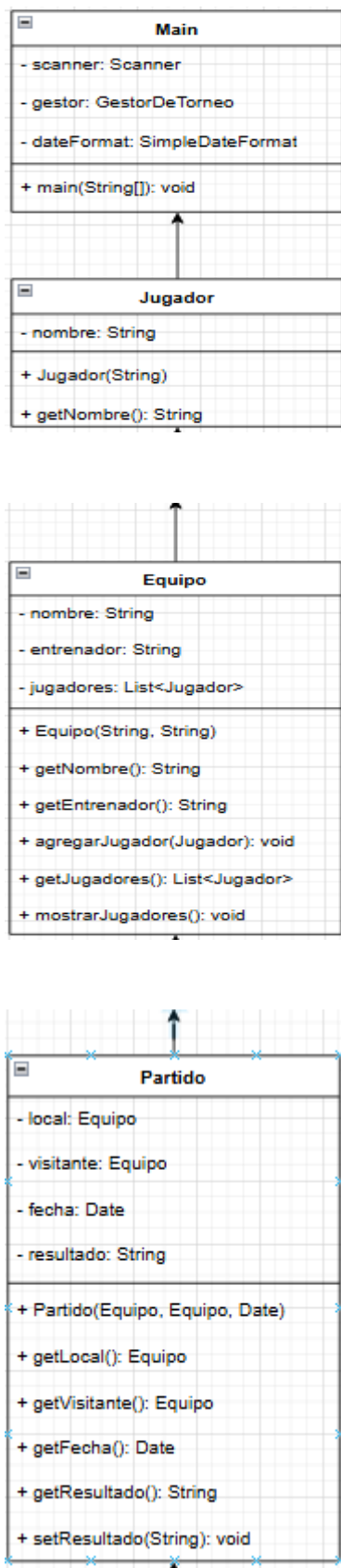
4. MARCO TEÓRICO

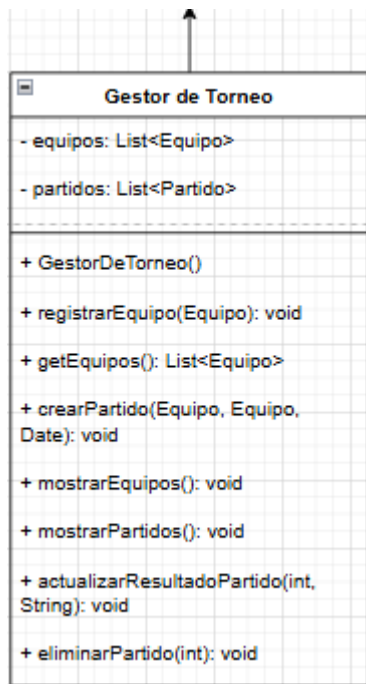
Implemente un sistema de gestión de torneos de fútbol utilizando Programación Orientada a Objetos (POO). El sistema debe permitir las siguientes funcionalidades:

- **Registrar Equipos:** o Cada equipo debe tener un nombre, un entrenador, y una lista de jugadores.
- **Crear Partidos:** o Registrar los equipos que jugarán un partido, la fecha del encuentro y el resultado final.
- **Consultar Información:** o Listar todos los equipos registrados con sus respectivos jugadores. o Mostrar el historial de partidos jugados, incluyendo los equipos participantes y los resultados.
- **Actualizar Resultados:** o Cambiar el estado de un partido de "Pendiente" a "Finalizado", añadiendo el marcador.

Actividad

DIAGRAMA UML:





CÓDIGO:

A continuación vamos a realizar una explicación de cada parte del código con la finalidad de saber como esta estructurada cada una de sus partes.

MAIN.JAVA

```

Main.java : Jugador.java : Equipo.java : Partido.java : GestorDeTorneo.java :
1 import java.text.ParseException;
2 import java.text.SimpleDateFormat;
3 import java.util.Date;
4 import java.util.Scanner;
5
6 public class Main {
7     public static void main(String[] args) {
8         Scanner scanner = new Scanner(System.in);
9         GestorDeTorneo gestor = new GestorDeTorneo();
10        SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
11
12        while (true) {
13            System.out.println("1. Registrar Equipo");
14            System.out.println("2. Agregar Jugador a Equipo");
15            System.out.println("3. Crear Partido");
16            System.out.println("4. Consultar Equipos");
17            System.out.println("5. Consultar Partidos");
18            System.out.println("6. Actualizar Resultado de Partido");
19            System.out.println("7. Salir");
20            System.out.print("Seleccione una opción: ");
21            int opcion = scanner.nextInt();
22            scanner.nextLine();
  
```

Importación de bibliotecas:

- **java.text.ParseException:** Para manejar errores al analizar fechas.
- **java.text.SimpleDateFormat:** Para definir el formato de las fechas.
- **java.util.Date:** Para trabajar con objetos de fecha.
- **java.util.Scanner:** Para leer entradas del usuario desde la consola.

Definición de la clase principal:

- **public class Main:** Es la clase principal donde se ejecutará el programa.

Método main:

- **public static void main(String[] args):** Es el punto de entrada del programa.

Creación de objetos:

- **Scanner scanner = new Scanner(System.in):** Se utiliza para leer datos ingresados por el usuario.
- **GestorDeTorneo gestor = new GestorDeTorneo():** Instancia un gestor para manejar las operaciones del torneo.
- **SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy"):** Define el formato para manejar las fechas (día/mes/año).

Bucle infinito (while (true)):

- Muestra un menú de opciones al usuario repetidamente hasta que seleccione "Salir".
- **Opciones del menú:**
 - 1: Registrar un equipo.
 - 2: Agregar un jugador a un equipo existente.
 - 3: Crear un partido entre equipos.
 - 4: Consultar la lista de equipos registrados.
 - 5: Consultar los partidos jugados.
 - 6: Actualizar el resultado de un partido pendiente.
 - 7: Salir del programa.
- **Selección del usuario:**
 - `int opcion = scanner.nextInt()`: Lee la opción seleccionada por el usuario.
 - `scanner.nextLine()`: Limpia el búfer después de leer la entrada.

```
23
24 ~      switch (opcion) {
25 ~          case 1:
26 ~              System.out.print("Ingrese el nombre del equipo: ");
27 ~              String nombreEquipo = scanner.nextLine();
28 ~              System.out.print("Ingrese el nombre del entrenador: ");
29 ~              String entrenador = scanner.nextLine();
30 ~              Equipo equipo = new Equipo(nombreEquipo, entrenador);
31 ~              gestor.registrarEquipo(equipo);
32 ~              System.out.println("Equipo registrado.");
33 ~              break;
34 ~
35 ~          case 2:
36 ~              System.out.print("Ingrese el nombre del equipo: ");
37 ~              String nombreEquipoJugador = scanner.nextLine();
38 ~              Equipo equipoEncontrado = null;
39 ~              for (Equipo e : gestor.getEquipos()) {
40 ~                  if (e.getNombre().equalsIgnoreCase(nombreEquipoJugador)) {
41 ~                      equipoEncontrado = e;
42 ~                      break;
43 ~                  }
44 ~              }
```


Estructura switch:

- Evalúa el valor de la variable opcion (seleccionada por el usuario en el menú) y ejecuta el caso correspondiente.

Caso 1: Registrar un equipo

- **Muestra un mensaje al usuario:** "Ingrese el nombre del equipo".

Entrada del usuario:

- **String nombreEquipo = scanner.nextLine():** Captura el nombre del equipo.
- **String entrenador = scanner.nextLine():** Captura el nombre del entrenador.

Creación del equipo:

- **Equipo equipo = new Equipo(nombreEquipo, entrenador):** Crea una nueva instancia de la clase Equipo con el nombre y entrenador proporcionados.

Registro del equipo:

- **gestor.registrarEquipo(equipo):** Llama al método del gestor para registrar el equipo en el sistema.

Confirmación:

- **System.out.println("Equipo registrado.):** Notifica al usuario que el equipo ha sido registrado exitosamente.
- **break:** Finaliza la ejecución de este caso.

Caso 2: Agregar un jugador a un equipo

Muestra un mensaje al usuario: "Ingrese el nombre del equipo".

Entrada del usuario:

- **String nombreEquipoJugador = scanner.nextLine():** Captura el nombre del equipo donde se desea agregar el jugador.

Búsqueda del equipo:

- **Equipo equipoEncontrado = null:** Inicializa una variable para almacenar el equipo encontrado.
- **for (Equipo e : gestor.getEquipos()):** Recorre la lista de equipos registrados en el gestor.
- **if (e.getNombre().equalsIgnoreCase(nombreEquipoJugador)):** Compara el nombre del equipo proporcionado por el usuario con el nombre de cada equipo en la lista, ignorando mayúsculas y minúsculas.

```
45 -         if (equipoEncontrado != null) {
46 -             System.out.print("Ingrese el nombre del jugador: ");
47 -             String nombreJugador = scanner.nextLine();
48 -             Jugador jugador = new Jugador(nombreJugador);
49 -             equipoEncontrado.agregarJugador(jugador);
50 -             System.out.println("Jugador agregado al equipo.");
51 -         } else {
52 -             System.out.println("Equipo no encontrado.");
53 -         }
54 -         break;
55 -
56 -     case 3:
57 -         System.out.print("Ingrese el nombre del equipo local: ");
58 -         String nombreLocal = scanner.nextLine();
59 -         System.out.print("Ingrese el nombre del equipo visitante: ");
60 -         String nombreVisitante = scanner.nextLine();
61 -         System.out.print("Ingrese la fecha del partido (dd/MM/yyyy): ");
62 -         String fechaString = scanner.nextLine();
63 -         Date fecha;
64 -         try {
65 -             fecha = dateFormat.parse(fechaString);
66 -         } catch (ParseException e) {
67 -             System.out.println("Formato de fecha inválido. Intente de nuevo.");
68 -             break;
```

Importación de bibliotecas:

- **java.util.Scanner:** Utilizada para capturar la entrada de datos desde el teclado.
- **java.text.SimpleDateFormat:** Define y maneja el formato para las fechas.
- **java.text.ParseException:** Maneja errores cuando el formato de la fecha ingresada no coincide con el esperado.

Primera parte: Agregar un jugador a un equipo

- **Condición:** Se verifica si el equipo (equipoEncontrado) existe. Si no es null, continúa con el proceso.
- **Entrada:** Solicita al usuario el nombre del jugador.

Acción:

- Se crea un objeto Jugador con el nombre ingresado.
- Este jugador es agregado al equipo encontrado mediante el método agregarJugador.

Salida:

- Si el jugador fue agregado, muestra un mensaje confirmándolo.
- Si el equipo no existe (null), muestra un mensaje indicando que el equipo no fue encontrado.

Segunda parte: Registrar un partido

Entradas:

- Solicita al usuario el nombre del equipo local.
- Solicita el nombre del equipo visitante.
- Solicita la fecha del partido en el formato dd/MM/yyyy.

Acción:

- La fecha ingresada se almacena como texto.
- Se intenta convertir esta fecha de texto a un objeto Date utilizando el formato especificado por dateFormat.
- Si el formato es incorrecto, se lanza una excepción ParseException.

Salida:

- Si la fecha es válida, continúa (aunque la lógica posterior no se muestra en la imagen).
- Si la fecha es inválida, muestra un mensaje indicando que el formato de la fecha es incorrecto.

```

69     }
70     Equipo local = null, visitante = null;
71     for (Equipo e : gestor.getEquipos()) {
72         if (e.getNombre().equalsIgnoreCase(nombreLocal)) {
73             local = e;
74         }
75         if (e.getNombre().equalsIgnoreCase(nombreVisitante)) {
76             visitante = e;
77         }
78     }
79     if (local != null && visitante != null) {
80         gestor.crearPartido(local, visitante, fecha);
81         System.out.println("Partido creado.");
82     } else {
83         System.out.println("Uno o ambos equipos no encontrados.");
84     }
85     break;
86
87     case 4:
88         gestor.mostrarEquipos();
89         break;
90

```

Equipo local = null, visitante = null;

- **Propósito:** Declara dos variables (local y visitante) de tipo Equipo y las inicializa con null, ya que al principio no se han asignado equipos a estas variables.

for (Equipo e : gestor.getEquipos()) {

- **Propósito:** Comienza un bucle que recorre todos los equipos disponibles, obtenidos mediante el método gestor.getEquipos(). Cada vez que pasa por el bucle, la variable e representa un equipo.
- **Explicación:** gestor.getEquipos() es probablemente una lista o conjunto de equipos que el gestor tiene registrados.

if (e.getNombre().equalsIgnoreCase(nombreLocal)) {

- **Propósito:** Verifica si el nombre del equipo actual (e.getNombre()) es igual al nombre del equipo local (nombreLocal), sin importar si se escribe en mayúsculas o minúsculas.

- **Explicación:** equalsIgnoreCase() compara dos cadenas de texto sin diferenciar entre mayúsculas y minúsculas. Si los nombres coinciden, el equipo actual es asignado a la variable local.

local = e;

- **Propósito:** Si el equipo encontrado es el equipo local, se asigna este equipo a la variable local.
- **Explicación:** La variable local ahora contiene el objeto de equipo que corresponde al equipo local que el usuario ha proporcionado.

if (e.getNombre().equalsIgnoreCase(nombreVisitante)) {

- **Propósito:** De forma similar al caso anterior, verifica si el nombre del equipo actual (e.getNombre()) coincide con el nombre del equipo visitante (nombreVisitante), sin importar mayúsculas o minúsculas.
- **Explicación:** Compara el nombre del equipo con el nombre del visitante. Si coinciden, el equipo actual se asigna a la variable visitante.

visitante = e;

- **Propósito:** Si el equipo encontrado es el equipo visitante, se asigna este equipo a la variable visitante.
- **Explicación:** La variable visitante ahora contiene el objeto de equipo que corresponde al equipo visitante que el usuario ha proporcionado.

}

- **Propósito:** Cierra el bloque de código del for. Ya se han comprobado todos los equipos para buscar al local y al visitante.

if (local != null && visitante != null) {

- **Propósito:** Verifica si ambos equipos, local y visitante, no son null. Esto significa que ambos equipos fueron encontrados correctamente.
- **Explicación:** Si ambas variables contienen un objeto Equipo válido (es decir, que no es null), entonces los equipos son válidos y se puede proceder a crear el partido.

gestor.crearPartido(local, visitante, fecha);

- **Propósito:** Llama al método crearPartido() del objeto gestor para crear un partido entre los equipos local y visitante, utilizando la fecha proporcionada.
- **Explicación:** El método crearPartido() probablemente registra el partido en algún sistema o base de datos de partidos, utilizando los dos equipos y la fecha como parámetros.

System.out.println("Partido creado.");

- **Propósito:** Imprime el mensaje "Partido creado." en la consola, indicando que el partido entre los dos equipos fue creado correctamente.
- **Explicación:** El usuario recibe una confirmación de que el partido ha sido creado.

else

- **Propósito:** Si uno o ambos equipos no fueron encontrados (es decir, alguno de los dos es null), se entra en este bloque de código.

- **Explicación:** Si la condición local != null && visitante != null no se cumple, significa que uno o ambos equipos no se encontraron.

System.out.println("Uno o ambos equipos no encontrados.");

- **Propósito:** Si no se encontró uno o ambos equipos, se muestra el mensaje "Uno o ambos equipos no encontrados." en la consola.
- **Explicación:** Informa al usuario que los equipos proporcionados no fueron encontrados en la base de datos o lista de equipos.

break;

- **Propósito:** Termina el bloque de código en el switch o bucle en el que se encuentra.
- **Explicación:** Este break indica que la ejecución del caso o bloque de código actual ha terminado, y el flujo de control continúa después del switch o bucle.

gestor.mostrarEquipos();

- **Propósito:** Llama al método mostrarEquipos() del objeto gestor para mostrar todos los equipos registrados.
- **Explicación:** Este método probablemente muestra la lista de equipos registrados en el sistema, permitiendo al usuario ver qué equipos están disponibles.

break;

- **Propósito:** Termina la ejecución del bloque de código en el switch o bucle donde se encuentra este comando.
- **Explicación:** Después de mostrar los equipos, el flujo de control se detiene para este caso específico, y el código continúa con la siguiente instrucción fuera del switch o bucle.


```

91         case 5:
92             gestor.mostrarPartidos();
93             break;
94
95         case 6:
96             System.out.print("Ingrese el índice del partido a actualizar: ");
97             int index = scanner.nextInt();
98             scanner.nextLine(); // Limpiar el buffer
99             System.out.print("Ingrese el marcador (ej. 2-1): ");
100             String marcador = scanner.nextLine();
101             gestor.actualizarResultadoPartido(index - 1, marcador);
102             System.out.println("Resultado actualizado.");
103             break;
104
105         case 7:
106             System.out.println("Saliendo...");
107             scanner.close();
108             return;
109
110         default:
111             System.out.println("Opción no válida.");
112     }
113 }
114 }
115 }

```

Opción 5:

- Llama al método `gestor.mostrarPartidos()` para mostrar una lista de partidos almacenados.
- Finaliza la ejecución de este caso con `break`.

Opción 6:

- Solicita al usuario que ingrese el índice del partido que desea actualizar y guarda este valor en la variable `index`.
- Usa `scanner.nextLine()` para limpiar el buffer después de leer un número con `nextInt()`.
- Luego, solicita al usuario que ingrese el nuevo marcador del partido, el cual se guarda en la variable `marcador`.

- Llama al método `gestor.actualizarResultadoPartido(index - 1, marcador)` para actualizar el marcador en una lista (ajustando el índice ingresado por el usuario restando 1, ya que los índices en programación suelen comenzar en 0).
- Finalmente, imprime un mensaje confirmando que el marcador fue actualizado y finaliza con `break`.

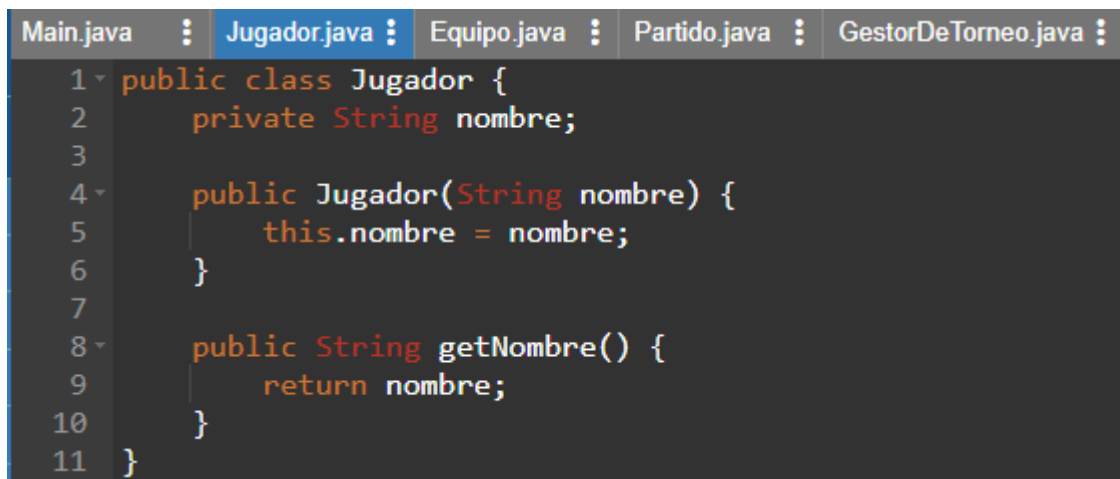
Opción 7:

- Muestra un mensaje indicando que el programa está saliendo.
- Cierra el objeto `Scanner` con `scanner.close()` para liberar recursos asociados.
- Utiliza `return` para finalizar la ejecución del programa.

Por defecto:

- Se ejecuta si el usuario ingresa una opción no válida.
- Muestra un mensaje de error indicando "Opción no válida".

JUGADOR.JAVA



```
1 public class Jugador {
2     private String nombre;
3
4     public Jugador(String nombre) {
5         this.nombre = nombre;
6     }
7
8     public String getNombre() {
9         return nombre;
10    }
11 }
```

Atributo nombre:

- Este atributo es de tipo String y se utiliza para almacenar el nombre del jugador.
- Está declarado como private, lo que significa que no puede ser accedido directamente desde fuera de la clase. Esto implementa el principio de encapsulación, asegurando que el atributo solo pueda ser modificado o leído a través de métodos específicos.

Constructor Jugador(String nombre):

- El constructor es público, lo que permite que se pueda crear un objeto Jugador desde cualquier parte del programa.
- Recibe un parámetro de tipo String, que representa el nombre del jugador.
- Dentro del constructor, se utiliza this.nombre para asignar el valor del parámetro al atributo de la clase. El uso de this diferencia entre el atributo de la clase y el parámetro del método, que tienen el mismo nombre.
- Este constructor garantiza que cada objeto Jugador se cree con un nombre específico.

Método getNombre():

- Este es un método público que devuelve el valor del atributo nombre.
- Su propósito es proporcionar acceso controlado al nombre del jugador, sin permitir que el atributo sea modificado directamente desde fuera de la clase.
- Al usar este método, otras clases pueden obtener el nombre del jugador de forma segura, cumpliendo con el principio de encapsulación.

EQUIPO.JAVA

```
Main.java  : Jugador.java  : Equipo.java  : Partido.java  : GestorDeTorneo.java  :
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class Equipo {
5      private String nombre;
6      private String entrenador;
7      private List<Jugador> jugadores;
8
9      public Equipo(String nombre, String entrenador) {
10         this.nombre = nombre;
11         this.entrenador = entrenador;
12         this.jugadores = new ArrayList<>();
13     }
14
15     public String getNombre() {
16         return nombre;
17     }
18
19     public String getEntrenador() {
20         return entrenador;
21     }
22
23     public void agregarJugador(Jugador jugador) {
24         jugadores.add(jugador);
25     }
```

Atributo nombre

- Este atributo es de tipo String y se utiliza para almacenar el nombre del equipo.
- Está declarado como private, lo que asegura que solo pueda ser accedido o modificado a través de los métodos de la clase, respetando el principio de encapsulación.

Atributo entrenador

- También es un atributo de tipo String que almacena el nombre del entrenador del equipo.
- Al igual que nombre, tiene el modificador private, por lo que no es accesible directamente desde fuera de la clase.

Atributo jugadores

- Este atributo es una lista de objetos de tipo Jugador y se declara como:

Ejemplo del code

java

Copiar código

```
private List<Jugador> jugadores;
```

- Utiliza el tipo genérico List para almacenar múltiples jugadores en el equipo. Inicialmente, la lista estará vacía.

Constructor Equipo(String nombre, String entrenador)

Este constructor inicializa los atributos de la clase:

- nombre se asigna usando el parámetro correspondiente.
- entrenador se asigna de manera similar.
- jugadores se inicializa como una nueva instancia de ArrayList < >, lo que significa que la lista comienza vacía y puede ser llenada posteriormente.
- El uso de this asegura que no haya ambigüedad entre los atributos de la clase y los parámetros recibidos.

Método getNombre()

- Método público que devuelve el valor del atributo nombre.
- Proporciona acceso controlado al nombre del equipo sin permitir modificarlo directamente.

Método getEntrenador()

- Similar a getNombre(), este método devuelve el valor del atributo entrenador.
- Permite obtener el nombre del entrenador del equipo de forma segura.

Método agregarJugador(Jugador jugador)

- Este método recibe un objeto de tipo Jugador como parámetro.
- Agrega el jugador recibido a la lista jugadores usando el método add() del ArrayList.
- Esto permite añadir jugadores al equipo de manera dinámica.

```

26
27 public List<Jugador> getJugadores() {
28     return jugadores;
29 }
30
31 public void mostrarJugadores() {
32     System.out.print("Jugadores: ");
33     for (Jugador jugador : jugadores) {
34         System.out.print(jugador.getNombre() + " ");
35     }
36     System.out.println();
37 }
38 }

```

Método getJugadores:

public List<Jugador> getJugadores():

- Este método devuelve una lista de jugadores almacenada en una variable llamada jugadores.
- La palabra clave List<Jugador> indica que se está manejando una lista de objetos de la clase Jugador.
- El acceso a este método es público, por lo que puede ser invocado desde otras clases.

return jugadores;:

- Devuelve la referencia a la lista de jugadores.

Método mostrarJugadores:

public void mostrarJugadores():

- Este método imprime en la consola la lista de jugadores.
- El tipo de retorno es void, lo que significa que no devuelve ningún valor.
- El acceso también es público.

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

- Muestra en la consola el texto "Jugadores: " antes de listar los nombres.

for (Jugador jugador : jugadores):

- Es un bucle for-each que recorre la lista jugadores.
- Por cada iteración, se toma un objeto de tipo Jugador de la lista y se almacena temporalmente en la variable jugador.

System.out.print(jugador.getNombre() + " ");:

- Para cada jugador, llama al método getNombre() (presumiblemente definido en la clase Jugador) para obtener el nombre del jugador.
- Imprime el nombre seguido de un espacio.

System.out.println();:

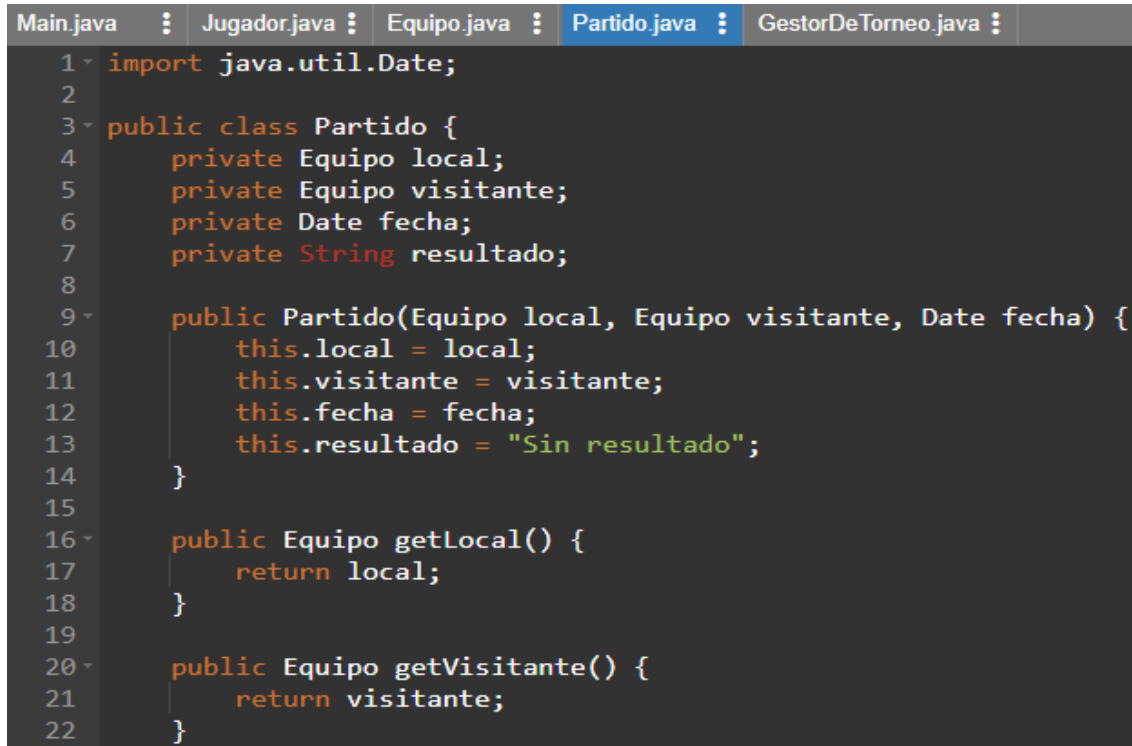
- Imprime una nueva línea para que el cursor baje a la siguiente línea después de listar todos los nombres.

Variables y objetos mencionados:

- **jugadores:**
 - Variable (probablemente un atributo de la clase) que contiene la lista de objetos Jugador.
 - Es de tipo List<Jugador>.
- **jugador:**
 - Variable temporal dentro del bucle for-each.
 - Representa cada objeto Jugador de la lista jugadores.

- **getNombre():**
 - Método que probablemente pertenece a la clase Jugador.
 - Devuelve el nombre del jugador como un String.

PARTIDO.JAVA



```
1 import java.util.Date;
2
3 public class Partido {
4     private Equipo local;
5     private Equipo visitante;
6     private Date fecha;
7     private String resultado;
8
9     public Partido(Equipo local, Equipo visitante, Date fecha) {
10         this.local = local;
11         this.visitante = visitante;
12         this.fecha = fecha;
13         this.resultado = "Sin resultado";
14     }
15
16     public Equipo getLocal() {
17         return local;
18     }
19
20     public Equipo getVisitante() {
21         return visitante;
22     }
```

Atributos (Variables de la clase):

local:

- Es un atributo privado que almacena el equipo local del partido.
- Su tipo es Equipo, que parece ser otra clase que representa un equipo.

visitante:

- Almacena el equipo visitante del partido.
- También es del tipo Equipo.

Fecha:

- Guarda la fecha en la que se jugará el partido.
- Es de tipo Date, que pertenece a la biblioteca estándar de Java (java.util.Date).

resultado:

- Almacena el resultado del partido como una cadena de texto (String).
- Inicialmente, este atributo tiene el valor "Sin resultado".

Constructor:

- Se utiliza para inicializar un partido.
- Recibe tres parámetros: el equipo local, el visitante y la fecha del partido.
- El resultado se asigna por defecto como "Sin resultado" al crearse un partido.

Métodos:**getLocal():**

- Permite obtener el equipo local del partido.
- Es un método público que devuelve el atributo local.

getVisitante():

- Permite obtener el equipo visitante.
- También es un método público que devuelve el atributo visitante.

```
23
24 ▾   public Date getFecha() {
25     |       return fecha;
26     |   }
27
28 ▾   public String getResultado() {
29     |       return resultado;
30     |   }
31
32 ▾   public void setResultado(String resultado) {
33     |       this.resultado = resultado;
34     |   }
35 }
```

getFecha():

- **Propósito:** Obtener la fecha del partido.
- **Retorno:** Devuelve el atributo fecha, que es de tipo Date.
- **Uso:** Sirve para consultar cuándo se llevará a cabo el partido.

getResultado():

- **Propósito:** Obtener el resultado del partido.
- **Retorno:** Devuelve el atributo resultado, que es de tipo String.
- **Uso:** Sirve para consultar cuál es el resultado actual del partido.

setResultado(String resultado):

- **Propósito:** Modificar el resultado del partido.
- **Parámetro:** Recibe un nuevo resultado como una cadena de texto (String).
- **Lógica:** Actualiza el atributo resultado con el valor proporcionado.
- **Uso:** Permite establecer un resultado final o intermedio para el partido.

GESTORDETORNEO.JAVA

```
Main.java : Jugador.java : Equipo.java : Partido.java : GestorDeTorneo.java :
1 import java.text.SimpleDateFormat;
2 import java.util.ArrayList;
3 import java.util.Date;
4 import java.util.List;
5
6 public class GestorDeTorneo {
7     private List<Equipo> equipos;
8     private List<Partido> partidos;
9
10    public GestorDeTorneo() {
11        equipos = new ArrayList<>();
12        partidos = new ArrayList<>();
13    }
14
15    public void registrarEquipo(Equipo equipo) {
16        equipos.add(equipo);
17    }
18
19    public List<Equipo> getEquipos() {
20        return equipos;
21    }
22
23    public void crearPartido(Equipo local, Equipo visitante, Date fecha) {
24        Partido partido = new Partido(local, visitante, fecha);
25        partidos.add(partido);
26    }
```

equipos:

- Tipo: List<Equipo> (lista de objetos de tipo Equipo).
- Modificador: private (solo accesible dentro de la clase GestorDeTorneo).
- Propósito: Almacena la lista de equipos registrados en el torneo.
- Inicialización: En el constructor, se inicializa como una nueva ArrayList.

partidos:

- Tipo: List<Partido> (lista de objetos de tipo Partido).
- Modificador: private (solo accesible dentro de la clase GestorDeTorneo).
- Propósito: Almacena la lista de partidos creados para el torneo.
- Inicialización: En el constructor, se inicializa como una nueva ArrayList.

Parámetros de los métodos:

- **equipo (en el método registrarEquipo):**
 - Tipo: Equipo.
 - Propósito: Es el equipo que se desea agregar a la lista equipos.
- **local, visitante, fecha (en el método crearPartido):**

local:

- Tipo: Equipo.
- Propósito: Representa el equipo local en un partido.

visitante:

- Tipo: Equipo.
- Propósito: Representa el equipo visitante en un partido.

fecha:

- Tipo: Date.
- Propósito: Indica la fecha en la que se jugará el partido.

```

27
28 ~   public void mostrarEquipos() {
29 ~       for (Equipo equipo : equipos) {
30 ~           System.out.println("Nombre: " + equipo.getNombre());
31 ~           System.out.println("Entrenador: " + equipo.getEntrenador());
32 ~           equipo.mostrarJugadores();
33 ~           System.out.println();
34 ~       }
35 ~   }
36
37 ~   public void mostrarPartidos() {
38 ~       SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
39 ~       for (int i = 0; i < partidos.size(); i++) {
40 ~           Partido partido = partidos.get(i);
41 ~           System.out.println("Partido " + (i + 1) + ": " +
42 ~                               partido.getLocal().getNombre() + " vs " +
43 ~                               partido.getVisitante().getNombre() + " | " +
44 ~                               dateFormat.format(partido.getFecha()) + " | " +
45 ~                               partido.getResultado());
46 ~       }
47 ~   }
48
49 ~   public void actualizarResultadoPartido(int index, String marcador) {
50 ~       if (index >= 0 && index < partidos.size()) {
51 ~           partidos.get(index).setResultado(marcador);
52 ~       } else {
53 ~           System.out.println("Índice inválido.");
54 ~       }
55 ~   }
56 ~ }

```

Método mostrarEquipos

1. equipo:

- **Tipo:** Equipo.
- **Alcance:** Declarada en el bucle for.
- **Propósito:** Representa cada equipo de la lista equipos durante la iteración.

Uso:

- Imprime el nombre del equipo (equipo.getNombre()).
- Imprime el nombre del entrenador (equipo.getEntrenador()).
- Llama al método mostrarJugadores() del equipo para listar a los jugadores.

Método mostrar Partidos

1. `dateFormat`:

- **Tipo:** SimpleDateFormat.
- **Propósito:** Formatea las fechas de los partidos en el formato dd/MM/yyyy.
- **Uso:** Para convertir las fechas de los partidos (`partido.getFecha()`) en una cadena legible.

2. `i`:

- **Tipo:** int.
- **Alcance:** Declarada en el bucle for.
- **Propósito:** Índice del bucle para recorrer la lista de partidos.

3. `partido`:

- **Tipo:** Partido.
- **Alcance:** Declarada en el bucle for.
- **Propósito:** Representa cada partido de la lista partidos durante la iteración.

Uso:

- Imprime los detalles de cada partido:
- Equipos locales y visitantes (`partido.getLocal().getNombre()` y `partido.getVisitante().getNombre()`).
- Fecha del partido (`dateFormat.format(partido.getFecha())`).
- Resultado del partido (`partido.getResultado()`).

Método actualizarResultadoPartido

index:

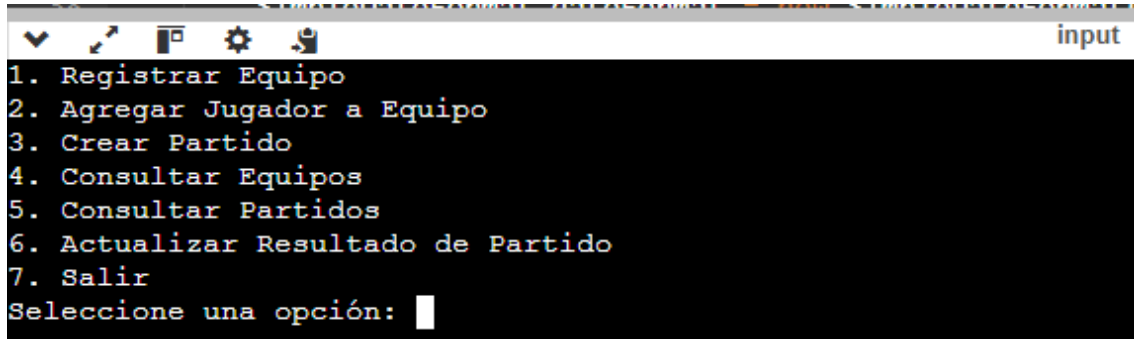
- **Tipo:** int.
- **Propósito:** Es el índice del partido en la lista partidos cuyo resultado se desea actualizar.
- **Uso:** Comprueba si el índice es válido y accede al partido correspondiente.

marcador:

- **Tipo:** String.
- **Propósito:** Contiene el resultado del partido que se desea actualizar.

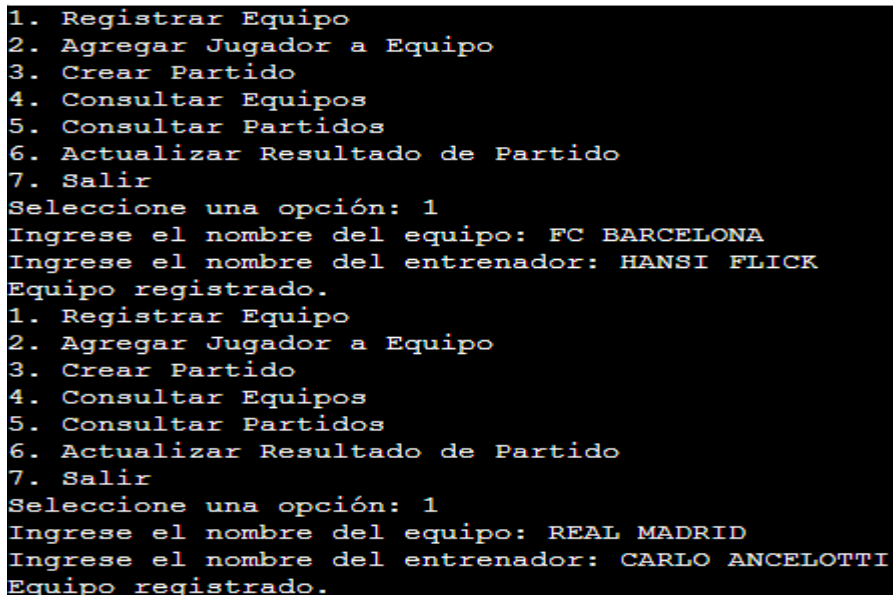
Ejecucion

A continuación, vamos a ver los códigos de ejecución del código que acabamos de observar y explicar anteriormente su función.



```
input
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: █
```

Esta ejecución presenta un menú interactivo para gestionar un torneo, donde el usuario puede registrar equipos, agregar jugadores a equipos, crear partidos, consultar equipos y partidos, actualizar el resultado de un partido, o salir del programa. Al ingresar el número de la opción deseada, el programa ejecutará la acción correspondiente.



```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 1
Ingrese el nombre del equipo: FC BARCELONA
Ingrese el nombre del entrenador: HANSI FLICK
Equipo registrado.
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 1
Ingrese el nombre del equipo: REAL MADRID
Ingrese el nombre del entrenador: CARLO ANCELOTTI
Equipo registrado.
```

En esta ejecución nos muestra un fragmento de la interfaz de un programa de consola que permite gestionar equipos de fútbol. En este caso, se registraron dos equipos: "FC Barcelona" con el entrenador "Hansi Flick" y "Real Madrid" con el entrenador "Carlo Ancelotti". Cada registro se realiza seleccionando la opción 1 del menú principal, que ofrece diversas funcionalidades, como agregar jugadores, crear partidos, consultar equipos o partidos, actualizar resultados y salir. Al ingresar los datos requeridos, el programa confirma que el equipo ha sido registrado exitosamente.

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 2
Ingrese el nombre del equipo: FC BARCELONA
Ingrese el nombre del jugador: TER STEGEN
Jugador agregado al equipo.
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 2
Ingrese el nombre del equipo: FC BARCELONA
Ingrese el nombre del jugador: ARAUJO
Jugador agregado al equipo.
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 2
Ingrese el nombre del equipo: FC BARCELONA
Ingrese el nombre del jugador: DANI OLMO
Jugador agregado al equipo.
```

En esta imagen podemos observar cómo se van agregando jugadores al equipo "FC Barcelona" usando un programa. Primero, se elige la opción de agregar jugadores (opción 2 del menú), y luego se escribe el nombre del equipo y el del jugador. En este caso, se añadieron tres jugadores: Ter Stegen, Araujo y Dani Olmo. Después de cada registro, el sistema confirma que el jugador fue añadido correctamente, mostrando un flujo claro y sencillo para completar la plantilla del equipo.

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 2
Ingrese el nombre del equipo: FC BARCELONA
Ingrese el nombre del jugador: LEWANDOWSKI
Jugador agregado al equipo.
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 2
Ingrese el nombre del equipo: REAL MADRID
Ingrese el nombre del jugador: COURTOIS
Jugador agregado al equipo.
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 2
Ingrese el nombre del equipo: REAL MADRID
Ingrese el nombre del jugador: ALABA
Jugador agregado al equipo.
```

En esta imagen se observa cómo se añaden jugadores a los equipos "FC Barcelona" y "Real Madrid" utilizando la opción 2 del menú del programa. Para el "FC Barcelona", se agregó a Lewandowski, mientras que para el "Real Madrid" se añadieron Courtois y Alaba. El procedimiento consiste en seleccionar la opción de agregar jugador, escribir el nombre del equipo y, a continuación, el nombre del jugador. Después de cada ingreso, el sistema confirma que el jugador fue añadido exitosamente, manteniendo un proceso claro y organizado.

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 2
Ingrese el nombre del equipo: REAL MADRID
Ingrese el nombre del jugador: MODRIC
Jugador agregado al equipo.
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 2
Ingrese el nombre del equipo: REAL MADRID
Ingrese el nombre del jugador: MBAPPE
Jugador agregado al equipo.
```

Este code permite administrar equipos de fútbol y realizar tareas como registrar equipos, agregar jugadores, crear partidos, y actualizar resultados. En la interacción que se muestra, alguien eligió agregar jugadores al equipo "REAL MADRID". Primero, añadió a "MODRIC", y luego repitió el proceso para "MBAPPE". En cada caso, el programa confirmó que el jugador fue agregado correctamente al equipo, de forma clara y sencilla. Es como una herramienta básica para gestionar un torneo o una liga.

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 3
Ingrese el nombre del equipo local: FC BARCELONA
Ingrese el nombre del equipo visitante: REAL MADRID
Ingrese la fecha del partido (dd/MM/yyyy): 27/10/2024
Partido creado.
```

Esta captura nos muestra cómo el programa permite crear un partido de fútbol. Se seleccionó la opción 3 (Crear Partido), y se ingresaron los datos necesarios: el equipo local, FC BARCELONA; el equipo visitante, REAL MADRID; y la fecha del partido, 27/10/2024. Finalmente, el sistema confirmó que el partido fue creado con éxito. Es un ejemplo de cómo el programa guía al usuario para gestionar encuentros de manera sencilla.

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 4
Nombre: FC BARCELONA
Entrenador: HANSI FLICK
Jugadores: TER STEGEN ARAUJO DANI OLMO LEWANDOWSKI

Nombre: REAL MADRID
Entrenador: CARLO ANCELOTTI
Jugadores: COURTOIS ALABA MODRIC MBAPPE
```

En esta interacción, el usuario seleccionó la opción 4 (Consultar Equipos) para obtener información sobre los equipos registrados. El programa muestra los datos de dos equipos: FC BARCELONA, dirigido por HANSI FLICK, con jugadores como TER STEGEN, ARAUJO, DANI OLMO y LEWANDOWSKI; y REAL MADRID, dirigido por CARLO ANCELOTTI, con jugadores como COURTOIS, ALABA, MODRIC y MBAPPE. Es una función útil para revisar los equipos, sus entrenadores y los jugadores asociados.

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 5
Partido 1: FC BARCELONA vs REAL MADRID | 27/10/2024 | Sin resultado
```

En esta ejecución nos muestra una interfaz de menú basada en texto para un programa de gestión deportiva. El usuario puede realizar diversas acciones, como registrar equipos, agregar jugadores a los equipos, crear partidos, consultar la lista de equipos y partidos, o actualizar los resultados de los encuentros. En este caso, el usuario seleccionó la opción 5, "Consultar Partidos", y el programa muestra un partido registrado entre el FC Barcelona y el Real Madrid programado para el 27 de octubre de 2024, pero que aún no tiene resultado.

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 6
Ingrese el índice del partido a actualizar: 1
Ingrese el marcador (ej. 2-1): 4-2
Resultado actualizado.
```

La ejecución nos muestra un menú similar al anterior, donde el usuario selecciona la opción 6, "Actualizar Resultado de Partido". El sistema solicita el índice del partido que se desea actualizar (en este caso, el partido 1) y luego pide ingresar el marcador en un formato específico, como "2-1". El usuario introduce el marcador "4-2", y el programa confirma que el resultado ha sido actualizado con éxito. Esto refleja un flujo interactivo para gestionar los resultados de partidos dentro del sistema.

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 5
Partido 1: FC BARCELONA vs REAL MADRID | 27/10/2024 | 4-2
```

En este caso nuevamente nos muestra el menú del programa, donde el usuario seleccionó la opción 5, "Consultar Partidos". Esta vez, el partido previamente registrado entre el FC Barcelona y el Real Madrid, programado para el 27 de octubre de 2024, ya incluye el resultado actualizado: 4-2. Esto indica que el marcador ingresado en la opción anterior (actualización de resultados) se reflejó correctamente en el listado de partidos.

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 7
Saliendo...
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

Es esta ejecución de consola nos ayuda a gestionar equipos y partidos deportivos a través de un menú interactivo. Las opciones incluyen registrar equipos, agregar jugadores a un equipo, crear y consultar partidos, además de actualizar los resultados de estos. En este caso, el usuario seleccionó la opción "Salir", que corresponde al número 7. El programa cerró su ejecución de forma correcta, mostrando el mensaje "Saliendo..." y terminando con el código de salida "0", lo que indica que todo funcionó sin problemas ni errores.

EN CASO DE NO ENCONTRAR EL EQUIPO:

```
1. Registrar Equipo
2. Agregar Jugador a Equipo
3. Crear Partido
4. Consultar Equipos
5. Consultar Partidos
6. Actualizar Resultado de Partido
7. Salir
Seleccione una opción: 2
Ingrese el nombre del equipo: MANCHESTER CITY
Equipo no encontrado.
```


En esta captura, el programa de consola muestra su menú principal para gestionar equipos y partidos. El usuario seleccionó la opción "2", que corresponde a "Agregar Jugador a Equipo". Luego, el programa pidió ingresar el nombre del equipo y el usuario escribió "MANCHESTER CITY". Sin embargo, el programa respondió con "Equipo no encontrado", lo que indica que el equipo no ha sido registrado previamente en el sistema, y, por lo tanto, no se puede agregar un jugador a este.

5. MATERIALES

Materiales digitales

- Onlinegdb
- Word

6. CONCLUSIONES

. El sistema de gestión debe implementar funcionalidades clave como registrar equipos, crear partidos y consultar información, todo basado en POO.

Es necesario incluir detalles completos de equipos, como jugadores y entrenador, para cumplir con los requisitos.

El sistema debe permitir actualizar el estado y los resultados de los partidos, pasando de "Pendiente" a "Finalizado" con el marcador correspondiente.

7. RECOMENDACIONES

Asegurarse de seguir un flujo lógico en el sistema, comenzando por registrar los equipos antes de intentar agregar jugadores o crear partidos.

Incluir validaciones y mensajes más claros para guiar al usuario y evitar errores, como mostrar una lista de equipos registrados al intentar agregar jugadores.

8. BIBLIOGRAFÍA

Fowler, M. (2003). *UML distilled: A brief guide to the standard object modeling language* (3rd ed.). Addison-Wesley.

Miles, R., & Hamilton, K. (2006). *Learning UML 2.0*. O'Reilly Media.

Martin, R. C. (2008). *Clean code: A handbook of agile software craftsmanship*.
Prentice Hall.

McConnell, S. (2004). *Code complete: A practical handbook of software construction* (2nd ed.). Microsoft Press.