



Proyecto grupal final

Luis Fernando Solano Montoya

Julio A. López Álvarez

Universidad Cenfotec

Estructura de datos I

Christian Sibaja Fernández

Tabla de Contenido

Capítulo 1- Descripción del problema	3
Capítulo 2 – Diagramas y TAD	4
UML	4
Lista	4
Cola	7
Capítulo 3 - Estado del aplicativo.	9
Capítulo 4 - Funcionamiento de la aplicación.....	10
Menú de inicio:	10
Opción - definir jugadores:	10
Opción - iniciar juego:	11
Opción - Mostrar puntaje.....	12
Opción – Mostrar récords ordenados:	13

Capítulo 1- Descripción del problema

Se nos ha contratado para crear una versión de un juego para computadora en el lenguaje C++. Se solicita realizar un análisis de los diferentes tipos de juegos y ver cuál de ellos puede cumplir los siguientes requisitos de juego:

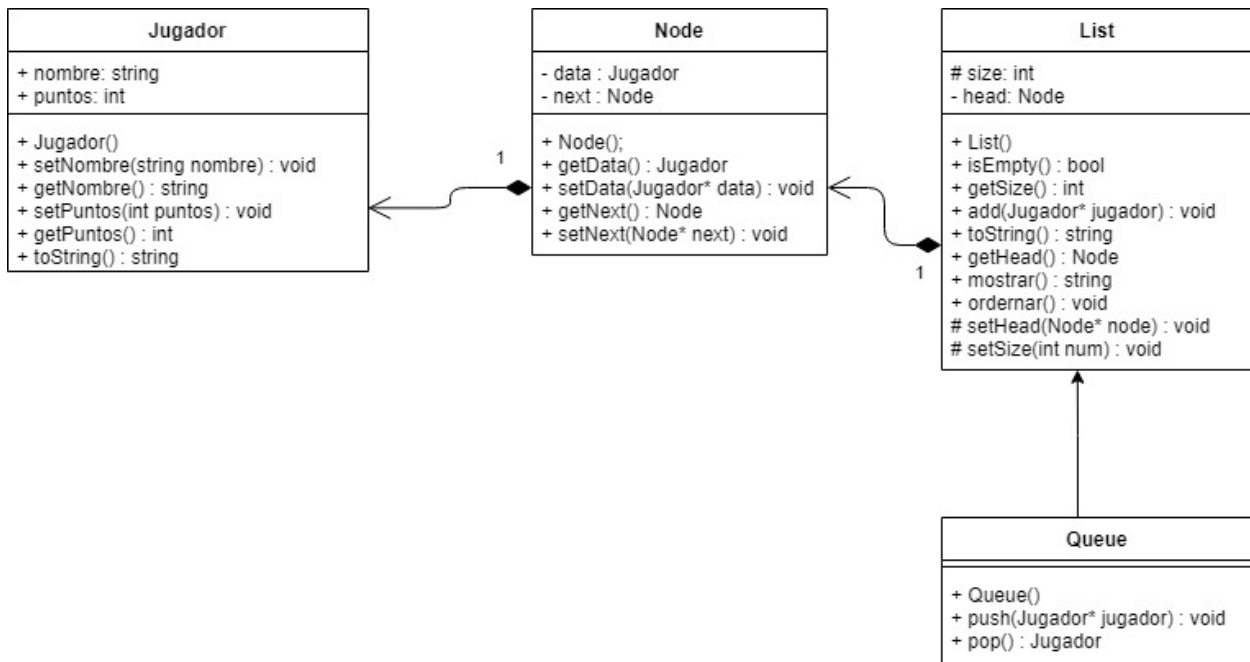
- a. Que permita al menos a dos jugadores interactuar por turnos
- b. Que, si se hacen varias partidas consecutivas, se pueda guardar el registro de los mejores jugadores (récores), según el tipo de juego (en archivos o base de datos).
- c. Debe contener acciones o información tal que permita trabajar con listas, colas, pilas (debe indicar dónde se utilizan). Lo importante es que ojalá estas estructuras sean parte integral de la lógica del programa y puedan darle soporte al juego interactuando entre sí.
- d. Se debe programar en capas y bajo el paradigma orientado a objetos.
- e. No es indispensable que sea un juego existente (Solitario, UNO, Gran Banco, etc.), se puede hacer una versión personalizada de alguno o inclusive a partir de una idea crear, pero es importante que sea funcional y permita ganar al menos a un jugador (mínimo 2 jugadores, máximo N).

Es por lo que para resolver este problema se decidió realizar el tradicional juego de piedra papel o tijera, donde hay una cola de jugadores a participar en el juego. Las reglas de negocio de este juego son:

- Piedra la gana a tijera.
- Tijera le gana a papel.
- Papel le gana piedra.

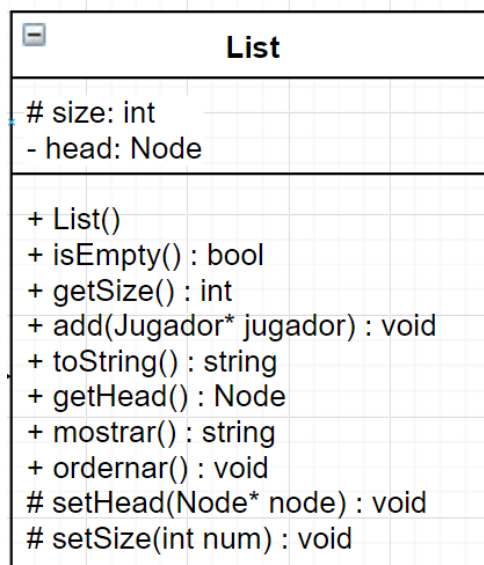
Capítulo 2 – Diagramas y TAD

UML



Lista

1. Representación abstracta



- a. Debe contener como atributos un nodo y un tamaño (size).
- b. Debe ser capaz de obtener y establecer sus atributos, validar si esta vacía, imprimir sus atributos en un método ToString() o mostrar y podrá ordenar sus nodos.

2. Invariante

- a. El nodo es de la clase Node, que contiene la información del jugador y puntos.
- b. Size será de tipo Integer, que contiene la cantidad de nodos que tiene.

3. Operaciones

List: (nada) -> Lista (Constructora)

// Construye un nuevo objeto List

isEmpty: nada -> booleano (Analizadora)

// Devuelve si la lista esta vacía

getSize: nada -> entero (Analizadora)

// Obtiene la cantidad de elementos de la lista

add: Jugador -> nada (Modificadora)

// Agrega un jugador a la lista

toString: nada -> string (Impresora)

// Obtiene la lista de elementos para guardar en txt

getHead: nada -> Node (Analizadora)

// Obtiene la cabeza de la lista

mostrar: nada -> string (Impresora)

// Muestra la lista de elementos para el usuario

ordenar: nada -> nada (Modificadora)

// Ordena los elementos de mayor a menor

setHead: Node -> nada (Modificadora)

// Modifica la cabeza de la lista

setSize: entero -> nada (Modificadora)

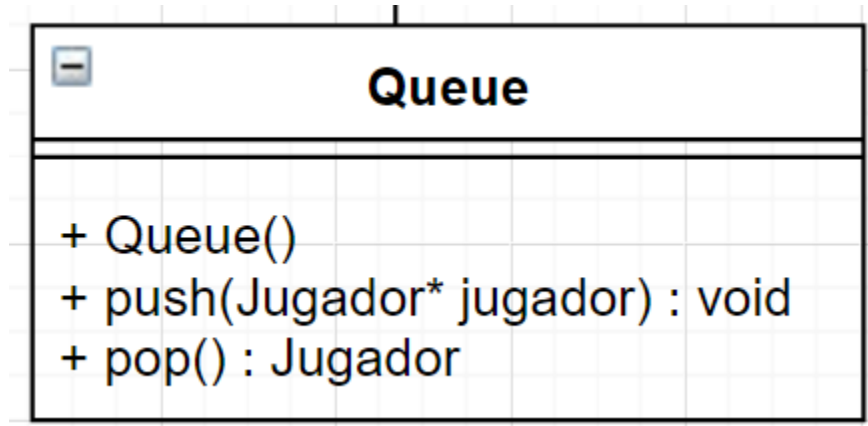
// Modifica el tamaño de la lista

4. Manejo del error

- a. Los errores se controlan en interfaz de UI pues solo se puede ejecutar las funciones desarrolladas por completo, por lo que en TAD de List no se hará ninguna validación.

Cola

1. Representación abstracta



- a. Esta clase implementa la clase List por lo que sus atributos son los mismos.
 - b. Debe ser capaz de atender y agregar nodos.
- ### 2. Invariante
- a. El nodo es de la clase Node, que contiene la información del jugador y puntos.
 - b. Size será de tipo Integer, que contiene la cantidad de nodos que tiene.
- ### 3. Operaciones

Queue: (nada) -> Queue (Constructora)

// Construye un nuevo objeto Queue

push: Jugador -> nada (Modificadora)

// Agrega a la cola un elemento

pop: nada -> Jugador (Analizadora)

// Elimina el primer elemento de la cola

4. Manejo del error

- a. Los errores se controlan en interfaz de UI pues solo se puede ejecutar las funciones desarrolladas por completo, por lo que en TAD de List no se hará ninguna validación.

Capítulo 3 - Estado del aplicativo.

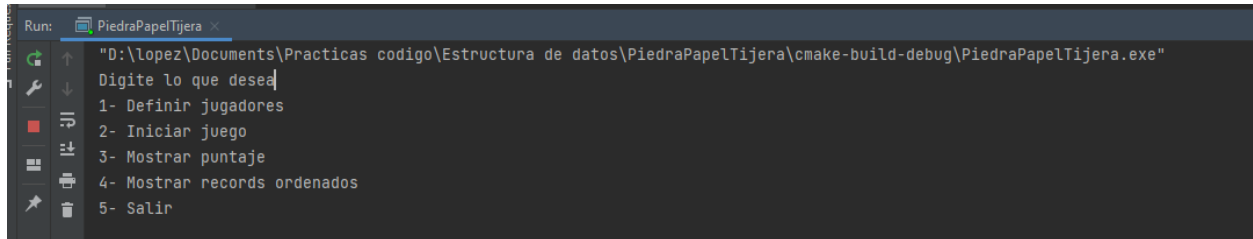
La aplicación es completamente funcional, cuenta con un menú que permite a los jugadores completar cualquiera de las siguientes acciones:

1. Definir jugadores
2. Iniciar juego
3. Mostrar puntaje
4. Mostrar récords ordenados
5. Salir

Capítulo 4 - Funcionamiento de la aplicación

A continuación, se muestra el funcionamiento de cada una de las funciones descritas en el capítulo 3. Al ser un aplicativo de consola, es precisamente esto lo que se mostrara, impresiones de la consola en ejecución.

Menú de inicio:



```
Run: PiedraPapelTijera x
"D:\lopez\Documents\Practicas codigo\Estructura de datos\PiedraPapelTijera\cmake-build-debug\PiedraPapelTijera.exe"
Digite lo que desea
1- Definir jugadores
2- Iniciar juego
3- Mostrar puntaje
4- Mostrar records ordenados
5- Salir
```

Opción - definir jugadores:



```
Run: PiedraPapelTijera x
1
Digite el nombre
jugador1
Digite lo que desea
1- Definir jugadores
2- Iniciar juego
3- Mostrar puntaje
4- Mostrar records ordenados
5- Salir
```

Opción - iniciar juego:

```

Digite lo que desea
1- Definir jugadores
2- Iniciar juego
3- Mostrar puntaje
4- Mostrar records ordenados
5- Salir
2
Juegan:
Jugador 1: Fer
Jugador 2: Jose
Elige jugador 1
Digite lo que jugador desea
1- Piedra
2- Papel
3- Tijera
1
Elige jugador 2
Digite lo que jugador desea
1- Piedra
2- Papel
3- Tijera
2
Jose es el ganador
Juegan:
Jugador 1: Julio
Jugador 2: Jose
Elige jugador 1
Digite lo que jugador desea
1- Piedra
2- Papel
3- Tijera
2
Elige jugador 2

```

```

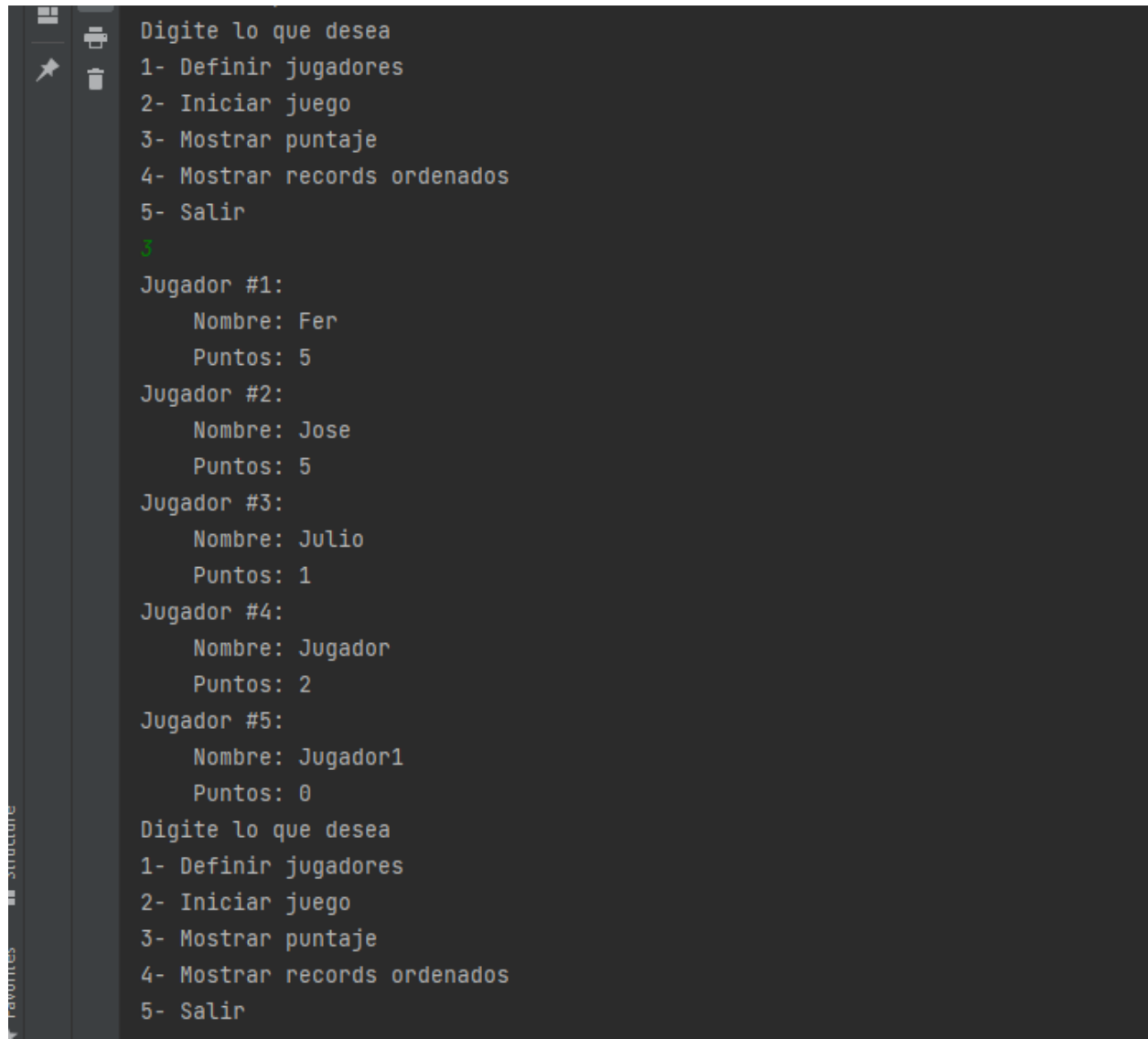
Elige jugador 2
Digite lo que jugador desea
1- Piedra
2- Papel
3- Tijera
3
Jose es el ganador
Juegan:
Jugador 1: Jugador
Jugador 2: Jose
Elige jugador 1
Digite lo que jugador desea
1- Piedra
2- Papel
3- Tijera
1
Elige jugador 2
Digite lo que jugador desea
1- Piedra
2- Papel
3- Tijera
3
Jugador es el ganador
Juegan:
Jugador 1: Jugador
Jugador 2: Jugador1
Elige jugador 1
Digite lo que jugador desea
1- Piedra
2- Papel
3- Tijera
3

```

```

Elige jugador 2
Digite lo que jugador desea
1- Piedra
2- Papel
3- Tijera
3
Jugador es el ganador
Fin de la partida.

```

Opción - Mostrar puntaje

```
Digite lo que desea
1- Definir jugadores
2- Iniciar juego
3- Mostrar puntaje
4- Mostrar records ordenados
5- Salir
3
Jugador #1:
  Nombre: Fer
  Puntos: 5
Jugador #2:
  Nombre: Jose
  Puntos: 5
Jugador #3:
  Nombre: Julio
  Puntos: 1
Jugador #4:
  Nombre: Jugador
  Puntos: 2
Jugador #5:
  Nombre: Jugador1
  Puntos: 0
Digite lo que desea
1- Definir jugadores
2- Iniciar juego
3- Mostrar puntaje
4- Mostrar records ordenados
5- Salir
```

Opción – Mostrar récords ordenados:

```
Digite lo que desea
1- Definir jugadores
2- Iniciar juego
3- Mostrar puntaje
4- Mostrar records ordenados
5- Salir
4
Jugador #1:
  Nombre: Fer
  Puntos: 5
Jugador #2:
  Nombre: Jose
  Puntos: 5
Jugador #3:
  Nombre: Jugador
  Puntos: 2
Jugador #4:
  Nombre: Julio
  Puntos: 1
Jugador #5:
  Nombre: Jugador1
  Puntos: 0
```