PECL1

SIMULACIÓN DEL CONTROL DE ACCESO A UN EVENTO DEPORTIVO

Alberto Larena Luengo - 09123282X

Alba Calvo Herrero - 03209702Y

22/11/2020

Índice

1. Detalles y justificación de la implementación

1.1. ESPECIFICACIÓN CONCRETA DE LA INTERFAZ DE LOS TAD'S IMPLEMENTADOS

1.1.1. TAD's creados:

espec PILA[AFICIONADO]

usa BOOLEANOS

parámetro formal

géneros aficionado

fparámetro

géneros pila

operaciones

{Generadoras}

pvacía: →pila {crea una pila vacía}

insertar: aficionado pila → pila

{Modificadora}

parcial **extraer**: pila → pila

{Observadoras}

contar: pila → natural

vacía?: pila → bool

ecuaciones de definitud

vacía?(p) = F ⇒ Def(extraer(p))

fespec

espec COLA[AFICIONADO]

usa BOOLEANOS

parámetro formal

géneros aficionado

fparámetro

géneros cola

operaciones

{Generadoras}

cvacía: → cola {crea una cola vacía}

insertar: aficionado cola → cola {insertar un aficionado en la cola}

{Modificadora}

parcial **eliminar**: cola → cola

{Observadoras}

contar: cola → natural

vacía?: cola → bool

var c: cola

ecuaciones de definitud

vacía?(c) = F ⇒ Def(eliminar(c))

fespec

espec LISTA[AFICIONADO]

usa LISTA[AFICIONADO], NATURALES2

operaciones

{Ver un dato en una posición, insertar, modificar o borrar}

parcial **_ [_]**: lista natural → aficionado

parcial **insertar**: aficionado lista natural → lista

parcial **modificar**: aficionado lista natural → lista

parcial **borrar**: lista natural → lista

{Ver si un dato está en la lista, y en qué posición se encuentra}

está?: aficionado lista → bool

buscar: aficionado lista → natural

var

n : natural

afi : aficionado

l : lista

ecuaciones de definitud

(1 ≤ i ≤ long(l)) = T ⇒ Def(l[i])

(1 ≤ i ≤ long(l)) = T ⇒ Def(modificar(afi,l,i))

(1 ≤ i ≤ long(l)) = T ⇒ Def(borrar(l,i))

(1 ≤ i ≤ long(l)+1) = T ⇒ Def(insertar(afi,l,i))

fespec

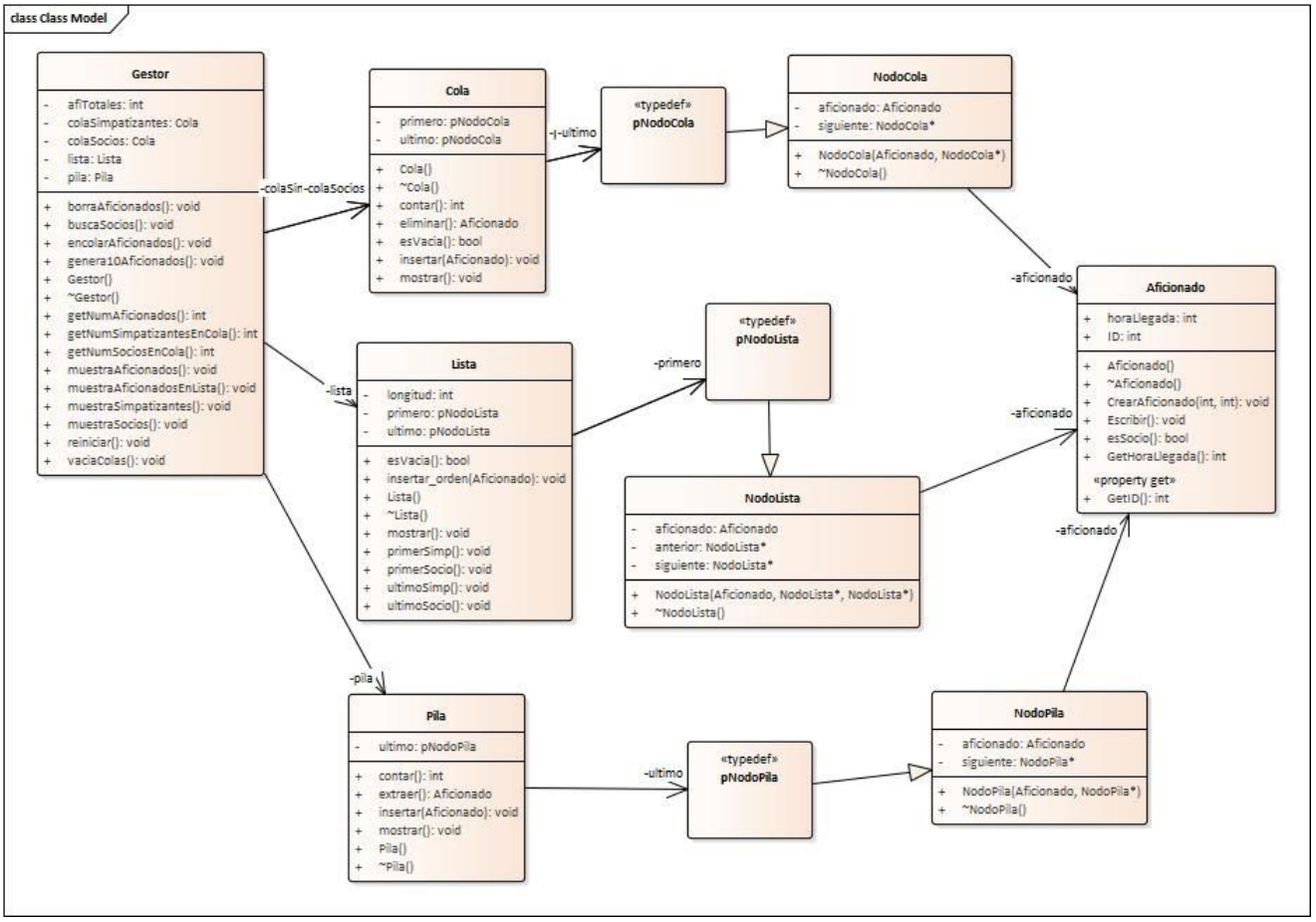
1.2.SOLUCIÓN ADOPTADA: DESCRIPCIÓN DE LAS DIFICULTADES ENCONTRADAS

A la hora de realizar la práctica no hemos encontrado muchas dificultades más allá de errores de sintaxis exceptuando el apartado de lista.

Nosotros optamos por realizar una lista doblemente enlazada porque para añadir directamente en orden resulta menos costosa, Cuando un aficionado es socio la recorrerá desde el primero hasta el último y cuando el aficionado no sea socio será desde el último hasta el primero, todo ello se hará con un puntero auxiliar aux. Dentro de la función insertar_orden a veces cuando hay muchos aficionados nos ocurre un fallo y es que al ejecutarla nos cierra el terminal, probando con el debugger encontramos que nos da fallo en la línea 32 con la instrucción: aux->anterior->siguiente = nuevo; Y comprobando los apuntes vemos que en la función insertar_orden de los apuntes se usa la misma instrucción y no hemos logrado averiguar a qué se debe este error.

1.3. DISEÑO DE LA RELACIÓN ENTRE LAS CLASES DE LOS TAD'S IMPLEMENTADOS

1.3.1. Diagrama UML



1.3.2. EXPLICACIÓN DE LOS MÉTODOS MÁS DESTACADOS

- *genera10Aficionados():*

  Genera con el ID y la hora de llegada aleatorios (ID entre 1 y 10 y la hora entre 0 y 59). Cada vez que se use la opción 0 el id tendrá un valor aleatorio entre 1 y 10, 2 y 20, 3 y 30 y así sucesivamente.

```
void Gestor::genera10Aficionados()
{
    srand(time(NULL));
    int id, num[10], h;
    Aficionado afi;
    for(int i = 0; i < 10; i++) {
        do {
            id = 1 + rand() % 10;
        } while(repeticion(id, num));
        num[i] = id;
        h = rand() % 60;
        afi.CrearAficionado(id + afiTotales, h);
        pila.insertar(afi);
    }
    afiTotales=afiTotales+10;
}
```

- *muestraAficionados():*

  Tiene como objetivo mostrar los 10 aficionados generados aleatoriamente que se encuentran en la pila. Por eso, llama a pila.mostrar() que sigue mostrando los aficionados de la pila hasta que último sea nulo, es decir, hasta que la pila esté vacía y no haya más aficionados que mostrar.

```
void Pila::mostrar()
{
    pNodoPila aux = ultimo;
    while(aux) {
        aux->aficionado.Escribir();
        aux = aux->siguiente;
    }
    cout << endl;
}
```

- *encolarAficionados()*:

  Introduce los aficionados en las colas. Comprueba uno a uno si es socio y si se cumple, lo introduce en colaSocios, sino, en colaSimpatizantes.

```
void Gestor::encolarAficionados()
{
    Aficionado afi;
    int nAficionados = getNumAficionados();
    for(int i = 0; i < nAficionados; i++) {
        afi = pila.extraer();
        if(afi.esSocio())
            colaSocios.insertar(afi);
        else
            colaSimpatizantes.insertar(afi);
    }
}
```

```
void Cola::insertar(Aficionado afi)
{
    pNodoCola nuevo;
    nuevo = new NodoCola(afi);
    if(ultimo)
        ultimo->siguiente = nuevo;
    ultimo = nuevo;
    if(!primero)
        primero = nuevo;
}
```

- *muestraSocios()*:

  Muestra los aficionados de colaSocios con el método colaSocios.mostrar(). Este los muestra hasta que primero sea nulo, es decir, hasta que la cola esté vacía.

```
void Cola::mostrar()
{
    pNodoCola aux=primero;
    while(aux){
        aux->aficionado.Escribir();
        aux = aux->siguiente;
        }
        cout << endl;
}
```

- *muestraSimpatizantes()*:

  Mismo funcionamiento que muestraSocios() pero con colaSimpatizantes.

- *muestraAficionadosEnLista()*:

```
void Gestor::muestraAficionadosEnLista()
{
    while(!colaSocios.esVacia()) {
        lista.insertar_orden(colaSocios.eliminar());
    }
    while(!colaSimpatizantes.esVacia()) {
        lista.insertar_orden(colaSimpatizantes.eliminar());
    }
    lista.mostrar();
}
```

  - *insertar_orden(): Con este método almacenaremos los aficionados de forma ordenada en función de la hora de llegada y priorizandolos si son socios*

```
void Lista::insertar_orden(Aficionado afi)
{
    pNodoLista nuevo, aux;
    nuevo = new NodoLista(afi);
    if(esVacia()) {
        primero = nuevo;
        ultimo = nuevo;
    } else if(afi.esSocio()) {
        if(afi.horaLlegada < primero->aficionado.horaLlegada) {
            nuevo->siguiente = primero;
            primero->anterior = nuevo;
            primero = nuevo;
        } else {
            aux = primero;
            while(aux != NULL && aux->aficionado.horaLlegada < afi.horaLlegada && aux->aficionado.esSocio()
                aux = aux->siguiente;
            }
            if(aux == NULL) {
                ultimo->siguiente = nuevo;
                nuevo->anterior = ultimo;
                ultimo = nuevo;
            } else {
                nuevo->siguiente = aux;
                aux->anterior->siguiente = nuevo;
                nuevo->anterior = aux->anterior;
                aux->anterior = nuevo;
            }
        }
    } else {

        if(afi.horaLlegada > ultimo->aficionado.horaLlegada || ultimo->aficionado.esSocio()) {
            nuevo->anterior = ultimo;
            ultimo->siguiente = nuevo;
            ultimo = nuevo;
        } else {
            aux = ultimo;
            while(aux->aficionado.horaLlegada > afi.horaLlegada && !aux->aficionado.esSocio()) {
                aux = aux->anterior;
            }
            nuevo->anterior = aux;
            aux->siguiente->anterior = nuevo;
            nuevo->siguiente = aux->siguiente;
            aux->siguiente = nuevo;
        }
    }

    longitud = longitud + 1;
}
```

- *buscaSocios(): En este método ejecutaremos los siguientes métodos de la clase lista:*
    - *primerSocio(): Muestra por pantalla el primer socio de la lista*
    - *ultimoSocio():Muestra por pantalla el último socio de la lista*
    - *primerSimp():Muestra por pantalla el primer simpatizante de la lista*
    - *ultimoSimp():Muestra por pantalla el último simpatizante de la lista*

```
void Lista::primerSocio()
{
    cout << "El primer socio es: " << setw(20);
    primero->aficionado.Escribir();
}
void Lista::ultimoSocio()
{
    pNodoLista aux = primero;
    while(aux->aficionado.esSocio()) {
        aux = aux->siguiente;
    }
    cout << "El último socio es: " << setw(20);
    aux->anterior->aficionado.Escribir();
}
void Lista::primerSimp()
{
    pNodoLista aux = primero;
    while(aux->aficionado.esSocio()) {
        aux = aux->siguiente;
    }
    cout << "El primer simpatizante es: " << setw(10);
    aux->aficionado.Escribir();
}
void Lista::ultimoSimp()
{
    cout << "El último simpatizante es: " << setw(10);
    ultimo->aficionado.Escribir();
}
```

1.4. EXPLICACIÓN DEL COMPORTAMIENTO DEL PROGRAMA

El programa simula cómo se controla el acceso de unos aficionados a un evento deportivo que empieza a las 19:00.

En el menú podemos visualizar en todo momento el número de aficionados en la pila y en las colas y debajo las opciones que se pueden realizar con el programa.

Primero, se generan 10 aficionados pulsando la opción **0**. Los aficionados tienen una hora de llegada (entre las 18:00 y las 18:59) y un identificador único. Ambos son generados aleatoriamente. En el caso del identificador, la primera vez que use esta opción será un número entre el 1 y el 10, la segunda entre el 11 y el 20, y así sucesivamente. Además, serán socios o simpatizantes según su identificador. (Si es par será socio y sino simpatizante).

Si pulsamos **1**, mostrará todos los aficionados que hay en la pila.

Pulsando **2**, borrará todos los aficionados almacenados en la pila.

Al pulsar la opción **3**, se almacenarán los aficionados en dos colas. Dependiendo de si son socios o no se insertarán en la cola de los socios o de los simpatizantes.

Con el **4** mostrará la cola de los aficionados socios y con el **5** la cola de los simpatizantes.

Si se pulsa la opción **6** se borrarán todos los aficionados almacenados en ambas colas.

La opción **7** extrae los aficionados de ambas colas y los almacena en una lista de forma ordenada. Primero, se extraerán los aficionados de la cola de los socios y se ordenarán en función de la hora de llegada de cada socio. Es decir, primero los que han llegado más pronto. Una vez hecho esto, se extraen los aficionados simpatizantes y también se ordenan según su hora de llegada.

Pulsando **8** buscará en la lista cuatro aficionados: el primer aficionado en acceder al estadio, el último socio en acceder al estadio y el último aficionado en acceder al estadio.

Si pulsamos **9** se reiniciará el programa al estado inicial.

Y por último si pulsamos **S** saldrá del programa.

1.5. BIBLIOGRAFÍA

Genera números aleatorios:

https://es.stackoverflow.com/questions/6040/rellenar-un-array-con-n%C3%BAmeros-aleatorios-sin-repetirse-los-n%C3%BAmeros

Para trabajar en equipo con github desktop:

https://www.youtube.com/watch?v=77W2JSL7-r8

Apuntes de teoría de la asignatura