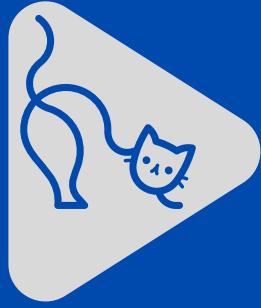


# MANUAL TECNICO



MOVIECATS  
STREAMING PLATAFORM

ESTRUCTURAS DE DATOS  
VACACIONES DE JUNIO, 2022  
FACULTAD DE INGENIERIA  
ESCUELA DE CIENCIAS Y SISTEMAS  
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

PROYECTO 2

Elaborado por  
Rodrigo Hernández  
201900042

```
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
```

```
.txtNew ul {padding-left:1.3em;list-style-type:none}
.txtNew ol {list-style-type:disc;}
.txtNew ol ol {list-style-type:decimal;}
.txtNew ol ul,.txtNew ol ol ul {list-style-type:circle}
.txtNew ol ol ol,.txtNew ol ol ol ul {list-style-type:square}
.txtNew ol [dir="rtl"],.txtNew ol ol [dir="rtl"] {
    padding-right:1.3em; margin-left:0; margin-right:1.3em}
.txtNew p {margin:0; line-height: normal; letter-spacing: normal}
.txtNew h1 {margin:0; line-height: normal; letter-spacing: normal}
.txtNew h2 {margin:0; line-height: normal; letter-spacing: normal}
.txtNew h3 {margin:0; line-height: normal; letter-spacing: normal}
.txtNew h4 {margin:0; line-height: normal; letter-spacing: normal}
.txtNew h5 {margin:0; line-height: normal; letter-spacing: normal}
.txtNew h6 {margin:0; line-height: normal; letter-spacing: normal}
.txtNew a {color: inherit;}</style><div data-packet="text" style="position: relative; width: 100%; height: 100%; overflow: hidden; border: 1px solid black; border-radius: 10px; background-color: #f0f0f0; font-family: sans-serif; font-size: 14px; color: black; padding: 10px; margin: auto; margin-top: 10px;">

<span class="color_2"><span style="font-size:18px; color:blue; font-weight:bold; margin-right:10px;">1380138113821383138413851386138713881389139013911392139313801381138213831384138513861387138813891390139113921393
```

# ÍNDICE

I.	Introducción .....	3
II.	Objetivo .....	3
III.	Dirigido .....	3
IV.	Especificación técnica.....	4
1.	Requisitos de Hardware.....	4
2.	Requisitos de Software.....	4
V.	Lógica de la Aplicación.....	5
1.	Clases .....	5
a.	Clase Película.....	5
b.	Clase Cliente.....	5
c.	Clase Actor.....	5
d.	Clase Categoría .....	6
e.	Clase Bloque .....	6
2.	Estructuras de Datos .....	7
a.	Árbol AVL.....	7
i.	Nodo AVL .....	7
ii.	Árbol AVL.....	7
b.	Lista Simple .....	9
i.	Nodo Cliente.....	9
ii.	Lista Simple .....	9
c.	Árbol Binario de Búsqueda.....	10
i.	Nodo ABB .....	10
ii.	Árbol ABB.....	10
d.	Tabla Hash .....	11
i.	Nodo Id Hash .....	11
ii.	Nodo Categoría.....	11
iii.	Tabla Hash.....	11

e.	Árbol de Merkle .....	12
i.	Nodo Cadena .....	12
ii.	Nodo Merkle .....	12
iii.	Árbol Merkle.....	13
f.	Blockchain.....	14
i.	Nodo Bloque .....	14
ii.	Blockchain.....	14
VI.	Diagramas de Clase .....	15
1.	Árbol AVL .....	15
2.	Lista Simple.....	15
3.	Árbol Binario de Búsqueda.....	16
4.	Tabla Hash .....	16
5.	Árbol de Merkle .....	17
6.	Blockchain .....	17
7.	Diagrama de todo el sistema .....	18

## I. Introducción

La aplicación web de alquiler de películas MovieCats fue realizado en el lenguaje de programación JavaScript junto con el apoyo de los lenguajes HTML y CSS donde la estructura visual fue apoyada por medio del uso de una plantilla donde se adaptó a los requerimientos del programa y de las estructuras de datos implementadas.

## II. Objetivo

El objetivo primordial del presente manual es orientar al programador acerca de la explicación de la forma en que fue creada la aplicación desde el código fuente, la ubicación de la aplicación de las estructuras de datos y la importante utilización de la Programación Orientada a Objetos en JavaScript para poder crear los distintos TDA e implementaciones de nuevas tecnologías como el Blockchain para la solución de los requerimientos del sistema.

## III. Dirigido

Este manual va orientado a los distintos programadores interesados en el conocimiento de las estructuras de datos y blockchain dentro de la programación web y la implementación del mismo con herramientas como D3 para mostrar las estructuras creadas en el lenguaje de programación.

## IV. Especificación técnica

### 1. Requisitos de Hardware

- Computadora de escritorio o portátil.
- Mínimo 4GB de Memoria RAM.
- Procesador Intel Core i3 o superior.
- Resolución gráfica máxima de 1900 x 1070 píxeles.

### 2. Requisitos de Software

- Sistema Operativo con que pueda contar con un navegador web y un Editor de Texto.
- Visual Studio Code u otro editor de texto.
- Librería D3.
- Graphviz.
- Navegador Web para el manejo de la aplicación web.

# V. Lógica de la Aplicación

## 1. Clases

### a. Clase Película

En esta clase será utilizada para almacenar objetos de tipo Película y poder interactuar con las estructuras de datos.

```
● ● ●

class Pelicula{

    constructor(_id,_nombre,_descripcion,_puntuacion,_precio){
        this.id_pelicula = _id
        this.nombre_pelicula = _nombre
        this.descripcion = _descripcion
        this.puntuacion_star = _puntuacion
        this.precio_Q = _precio
    }
}
```

### b. Clase Cliente

En esta clase será utilizada para almacenar objetos de tipo Cliente y poder interactuar con las estructuras de datos.

```
● ● ●

class Cliente{

    constructor(_dpi,_nombre,_username,_correo,_contrasenia,_telefono){
        this.dpi = _dpi
        this.nombre = _nombre
        this.username = _username
        this.correo = _correo
        this.contrasenia = _contrasenia
        this.telefono = _telefono
    }
}
```

### c. Clase Actor

En esta clase será utilizada para almacenar objetos de tipo Actor y poder interactuar con las estructuras de datos.

```
● ● ●  
  
class Actor{  
    constructor(_dni,_nombre,_correo,_descripcion){  
        this.dni = _dni  
        this.nombre = _nombre  
        this.correo = _correo  
        this.descripcion = _descripcion  
    }  
}
```

## d. Clase Categoría

En esta clase será utilizada para almacenar objetos de tipo Autor y poder interactuar con las estructuras de datos.

```
● ● ●  
  
class Categoria{  
    constructor(_id,_company){  
        this.id = _id  
        this.company = _company  
    }  
}
```

## e. Clase Bloque

En esta clase será utilizada para almacenar los datos del Bloque generado del Blockchain.

```
● ● ●  
  
class Bloque{  
    constructor(_index,_timestamp,_data,_nonce,_rootmerkle,_hash){  
        this.index = _index  
        this.timestamp = _timestamp  
        this.data = _data  
        this.nonce = _nonce  
        this.previoushash = "00"  
        this.rootmerkle = _rootmerkle  
        this.hash = _hash  
    }  
}
```

## 2. Estructuras de Datos

### a. Árbol AVL

Esta estructura es no lineal y este árbol en sus inserciones procede a hacer las respectivas rotaciones para poder balancearse.

#### i. Nodo AVL

Este nodo se encarga de almacenar una película y almacena un izquierda, un derecha y la altura del nodo.

```
● ● ●  
  
class NodoAVL{  
    constructor(_pelicula){  
        this.pelicula = _pelicula  
        this.izquierda = null  
        this.derecha = null  
        this.altura = 0  
    }  
}
```

#### ii. Árbol AVL

En esta clase se contendrá todas las operaciones para almacenar, mostrar, buscar y graficar películas.

```
● ● ●

class ArbolAVL{
    constructor(){
        this.raiz = null
        this.codigodot = ""
        this.mostrar = ""
        this.guardar = "["
    }
    insertar(_pelicula){
        //Llama a su metodo insertarrecursivo
    }
    insertarrecursivo(pelicula,nodo){
        //Procede a insertar la pelicula en el arbol
    }
    altura(nodo){
        //Calcula la altura del nodo
    }
    maximo(valor1,valor2){
        //Calcula cual es el mayor entre los dos valores
    }

    rotacionizquierda(nodo){
        //Cambia la posicion de los nodos rotando a la
        //izquierda
    }

    rotacionderecha(nodo){
        //Cambia la posicion de los nodos rotando a la
        //derecha
    }
    rotaciondoblederecha(nodo){
        //Cambia la posicion de los nodos rotando a la
        //izquierda y luego a la derecha
    }

    rotaciondobleizquierda(nodo){
        //Cambia la posicion de los nodos rotando a la
        //derecha y luego a la izquierda
    }

    preordenG(){
        //Llama a su metodo recursivo PreordenG
    }
    pre_ordenG(nodo){
        //Crea los nodos y conexiones del grafo para
        //graphviz
    }
    graficar(){
        //Llama al preordenG y une todo el codigo en .dot
        //para poder graficar el árbol.
    }

    inordenA(){
        //Llama a su metodo recursivo in_ordenA
    }
    in_ordenA(nodo){
        //Metodo para mostrar las peliculas de forma
        //Ascendente
    }
    inordenD(){
        //Llama a su metodo recursivo in_ordenD
    }
    in_ordenD(nodo){
        //Metodo para mostrar las peliculas de forma
        //Descendente
    }
    inorden(){
        //Llama a su metodo recursivo in_orden
    }
    in_orden(nodo){
        //Muestra en consola el metodo inorden
    }
    postorden(){
        //Llama a su metodo recursivo post_orden
    }
    post_orden(nodo){
        //Muestra en consola el metodo postorden
    }
    buscar(nodo,_id){
        //busca la pelicula en todo el arbol
    }
    preordenGuardar(){
        //Llama a su metodo recursivo pre_ordenGuardar
    }
    pre_ordenGuardar(nodo){
        //Guarda los nodos para hacer los cambios en las
        //peliculas.
    }
}
```

## b. Lista Simple

Esta estructura se implementa para poder guardar todos los Clientes almacenados en el sistema.

### i. Nodo Cliente

Este nodo se encarga de almacenar los objetos de tipo Cliente y su siguiente.

```
● ● ●

class NodoCliente{
    constructor(_cliente){
        this.cliente = _cliente
        this.siguiente = null
    }
}
```

### ii. Lista Simple

En esta clase se hacen las diferentes operaciones que requiere la lista simple para operar con los Clientes.

```
● ● ●

class ListaSimple{
    constructor(){
        this.primero = null
    }
    insertar(_cliente){
        //Inserta al cliente a la lista
    }
    validarcliente(username,password){
        //Valida si las credenciales son correctas
    }
    retornaruserlogin(username,password){
        //Retorna el Cliente por medio del username y password
    }
    retornaruser(_username){
        //Retorna el Cliente por medio del username
    }
    graficar(){
        //Grafica la estructura utilizando el codigop dot
    }
}
```

## c. Árbol Binario de Búsqueda

En esta estructura se enfoca en almacenar objetos de tipo Actor por medio de su dni.

### i. Nodo ABB

En esta clase se encarga almacenar al actor, a su nodo izquierdo y derecho.

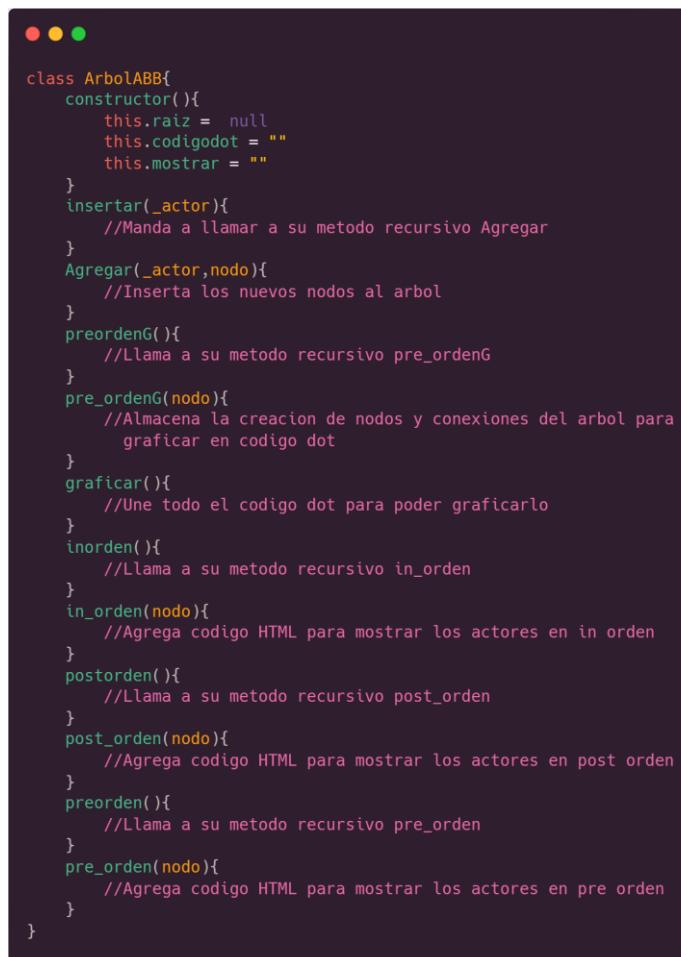


```
● ● ●

class NodoABB{
    constructor(_actor){
        this.actor = _actor
        this.izquierda = null
        this.derecha = null
    }
}
```

### ii. Árbol ABB

Esta clase se encarga de hacer las operaciones de los actores como insertar y graficar.



```
● ● ●

class ArbolABB{
    constructor(){
        this.raiz = null
        this.codigodot = ""
        this.mostrar = ""
    }
    insertar(_actor){
        //Manda a llamar a su metodo recursivo Agregar
    }
    Agregar(_actor,nodo){
        //Inserta los nuevos nodos al arbol
    }
    preorderG(){
        //Llama a su metodo recursivo pre_ordenG
    }
    pre_ordenG(nodo){
        //Almacena la creacion de nodos y conexiones del arbol para
        //graficar en codigo dot
    }
    graficar(){
        //Une todo el codigo dot para poder graficarlo
    }
    inorder(){
        //Llama a su metodo recursivo in_orden
    }
    in_orden(nodo){
        //Agrega codigo HTML para mostrar los actores en in orden
    }
    postorden(){
        //Llama a su metodo recursivo post_orden
    }
    post_orden(nodo){
        //Agrega codigo HTML para mostrar los actores en post orden
    }
    preorder(){
        //Llama a su metodo recursivo pre_orden
    }
    pre_orden(nodo){
        //Agrega codigo HTML para mostrar los actores en pre orden
    }
}
```

## d. Tabla Hash

En esta estructura es un tipo de tabla hash abierto donde se almacenan todas las categorías en el hash teniendo un método de inserción de  $\text{id} \bmod \text{len}(\text{hash})$  que inicialmente tiene de tamaño 20 y agregando tiene un método de rehashing de 5 posiciones pasando del 75% de su capacidad.

### i. Nodo Id Hash

En esta clase se encarga de guardar el id del nodo hash, la categoría, su siguiente y su derecho.

```
● ● ●  
  
class NodoIdHash{  
    constructor(_id){  
        this.id = _id  
        this.categoría = null  
        this.siguiente = null  
        this.derecho = null  
    }  
}
```

### ii. Nodo Categoría

En esta clase se encarga de guardar la categoría y su siguiente.

```
● ● ●  
  
class NodoCategoria{  
    constructor(categoría){  
        this.categoría = categoría  
        this.derecho = null  
    }  
}
```

### iii. Tabla Hash

En esta clase se encarga de crear la tabla Hash donde se almacenaran las categorías contando con sus métodos para manejar la tabla.

```

● ● ●

class TablaHash{
    constructor(){
        this.tamano = 0
        this.cabeza = null
        this.llenos = 0
        this.mostrar = ""
        this.llenadoinicial()
    }
    llenadoinicial(){
        //Llena inicialmente el hash con 20 casillas
    }
    insertar(categoría){
        //Inserta el nodo conforme al resultado entre id mod len(hash)
        //siempre y cuando no sobrepase el 75%
    }
    rehashing(){
        //Agrega 5 espacios mas al hash
    }
    graficar(){
        //Grafica la estructura de la tabla hash con codigo dot
    }
    mostrarhtml(){
        //Agrega codigo HTML de todas las categorias agregadas
    }
}

```

## e. Árbol de Merkle

Esta estructura está hecha para poder generar el hash padre que será utilizado para el Blockchain de las transacciones.

Cada transacción insertada al árbol cuenta con la siguiente estructura en sus datablocks:

"Nombre del Cliente - Nombre de la Película"

### i. Nodo Cadena

En este nodo se encarga de almacenar la cadena entrante.

```

● ● ●

class NodoCadena{
    constructor(cadena){
        this.cadena = cadena
    }
}

```

### ii. Nodo Merkle

En este nodo se encarga de almacenar el hash y su izquierda y derecha.

```

● ● ●

class NodoMerkle{
    constructor(hash){
        this.hash = hash
        this.izquierda = null
        this.derecha = null
    }
}

```

### iii. Árbol Merkle

Aquí se encuentran todos los métodos para operar el árbol.

```
● ● ●

class ArbolMerkle{
    constructor(){
        this.tophash = null
        this.iniciales = new Lista()
        this.codigodot = ""
    }
    insertar(cadena){
        //Agrega la cadena a los nodos iniciales(datablocks)
    }

    crearArbol(exp){
        //Comienza a crear el arbol a partir del tophash o nodo padre
    }

    crear_arbol(nodo, exp){
        //Crea los demas nodos del arbol recursivamente
    }
    generarhash(nodo,n){
        //Genera el hash y los asigna a los nodos usando el SHA256
    }
    preorden(){
        //Llama a su metodo recursivo pre_orden
    }
    pre_orden(nodo){
        //Visualiza en preorden el arbol en consola
    }
    auth(){
        /*
            Genera el arbol de tal manera que cumpla que sea de orden
            exponencial 2 y agregandole sus hash y llamando a sus
            metodos.
        */
    }
    preordenG(){
        //Llama a su metodo recursivo pre_ordenG
    }
    pre_ordenG(nodo, num){
        //Crea los nodos del arbol y sus conexiones en codigo dot
        //para poder graficarlo
    }
    graficar(){
        //Grafica el arbol utilizando el codigo dot
    }
}
```

## f. Blockchain

Esta estructura se encarga de almacenar como que fuera una lista simple los bloques del blockchain contando con ciertas restricciones:

```

TIMESTAMP: •Es la fecha y hora exacta en la que se creó el bloque. Debe de tener el siguiente formato: (DD-MM-YY--::HH:MM:SS).

DATA: •Contendrá todas las transacciones hechas antes de presionar el botón.

NONCE: •Será el número entero que se debe iterar de uno en uno hasta encontrar un hash que cumpla con la prueba de trabajo que tenga inicialmente 00.

PREVIOUSHASH: •Es el hash del bloque previo, este es necesario para validar que la cadena de bloques no esté corrupta. En caso del bloque génesis, el hash anterior debe de ser 00.

ROOTMERKLE: En este bloque se almacena el nodo padre del árbol de Merkle.

HASH (bloque actual): •El hash deberá generarse aplicando la función SHA256 a las propiedades: INDEX, TIMESTAMP, PREVIOUSHASH, ROOTMERKLE y NONCE, es decir SHA256(INDEX+TIMESTAMP+PREVIOUSHASH+ROOTMERKLE+NONCE). Para considerar el hash como válido este debe de tener un prefijo de dos ceros.

```

### i. Nodo Bloque

Objeto que almacena al usuario y la cantidad de libros pendientes.

```

class NodoBloque{
    constructor(_bloque){
        this.bloque = _bloque
        this.siguiente = null
    }
}

```

### ii. Blockchain

Clase donde se realizan todas las operaciones del blockchain como insertar y graficar.

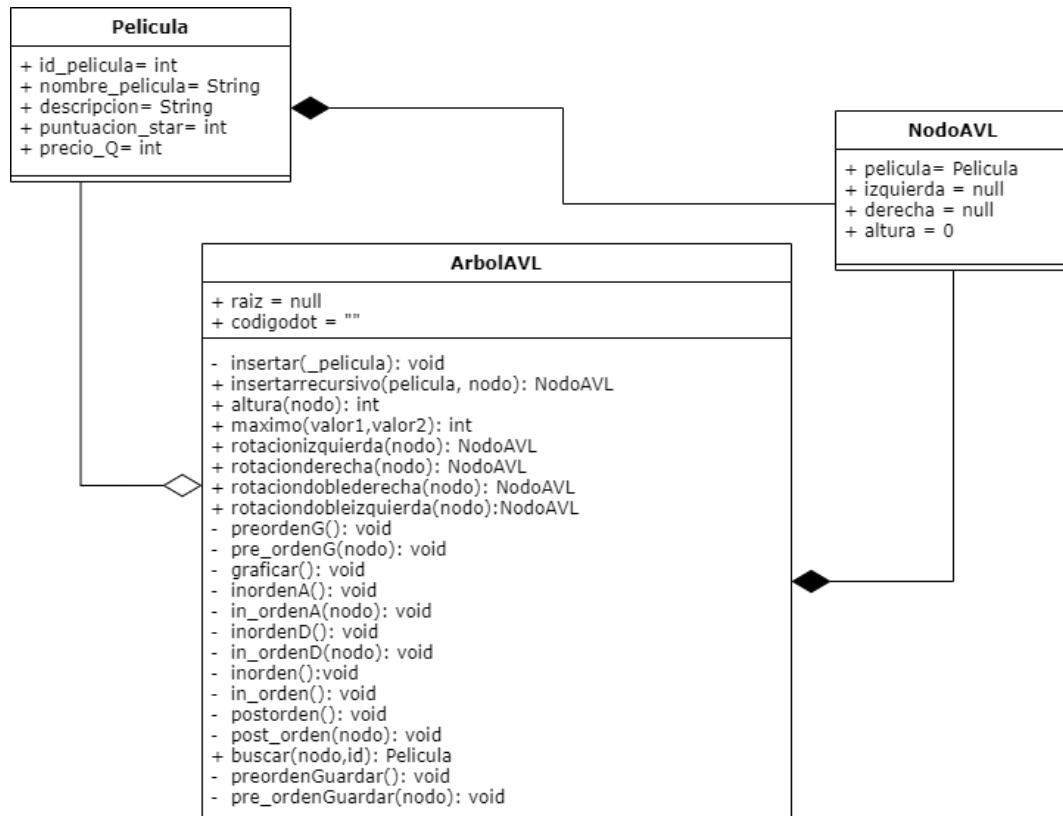
```

class Blockchain{
    constructor(){
        this.bloque_genesis = null
    }
    agregar(_bloque){
        //Metodo que agrega los bloques y verifica si es bloque genesis o no por su previous hash
    }
    graficar(){
        //Metodo para graficar la estructura del blockchain en código
        dot
    }
}

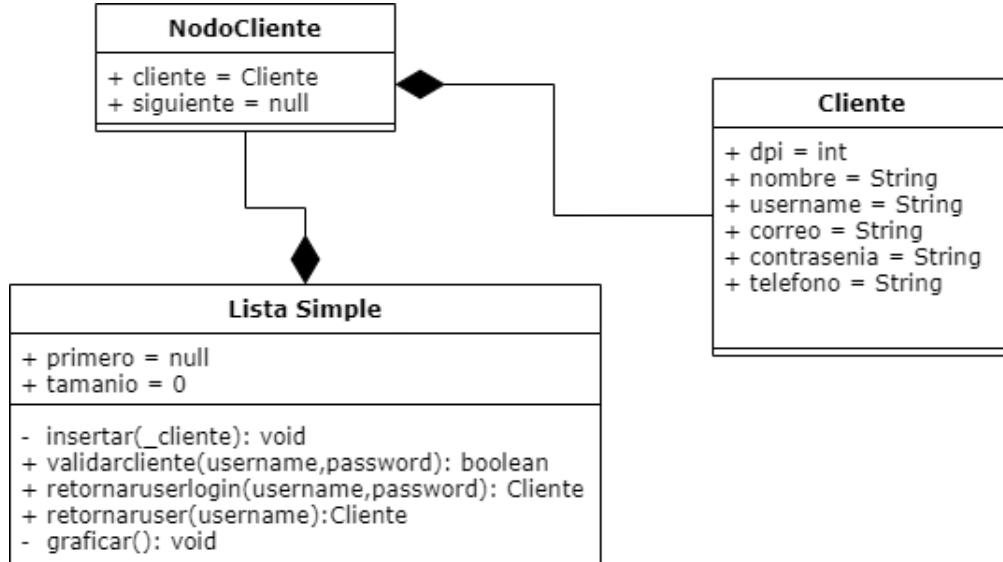
```

# VI. Diagramas de Clase

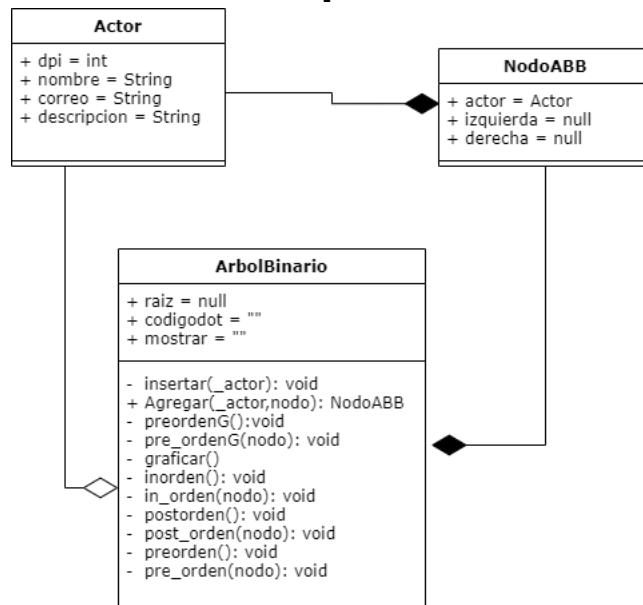
## 1. Árbol AVL



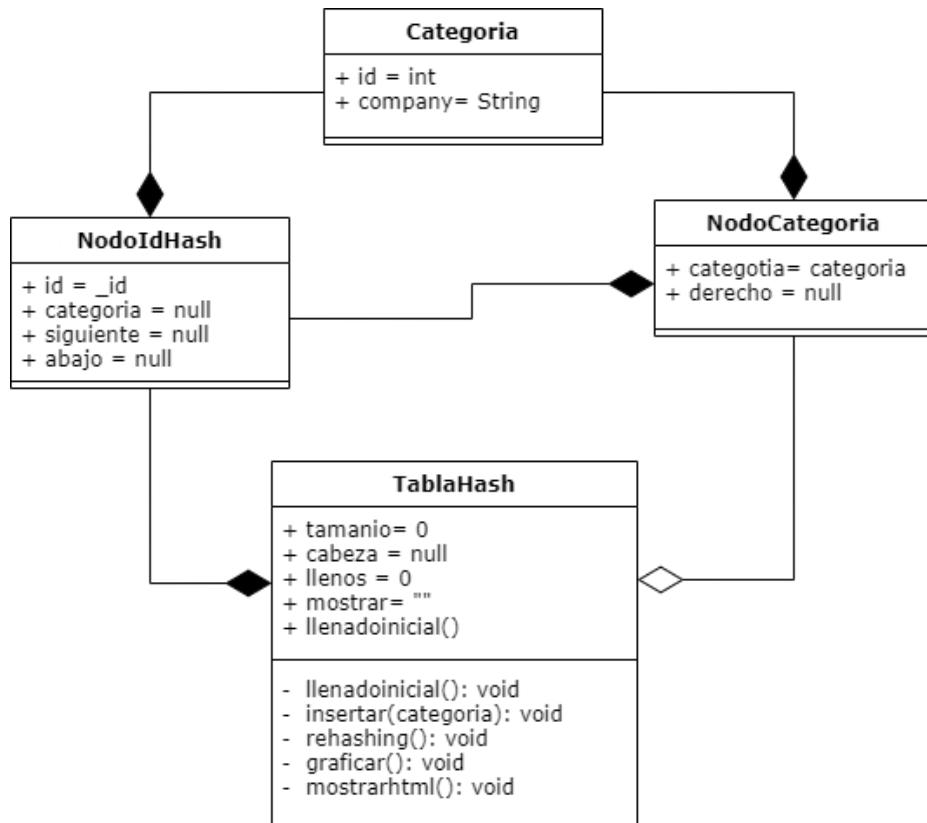
## 2. Lista Simple



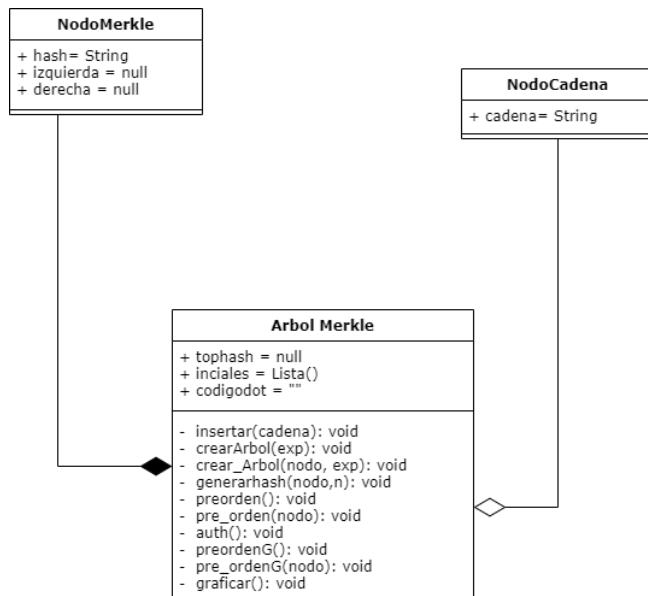
### 3. Árbol Binario de Búsqueda



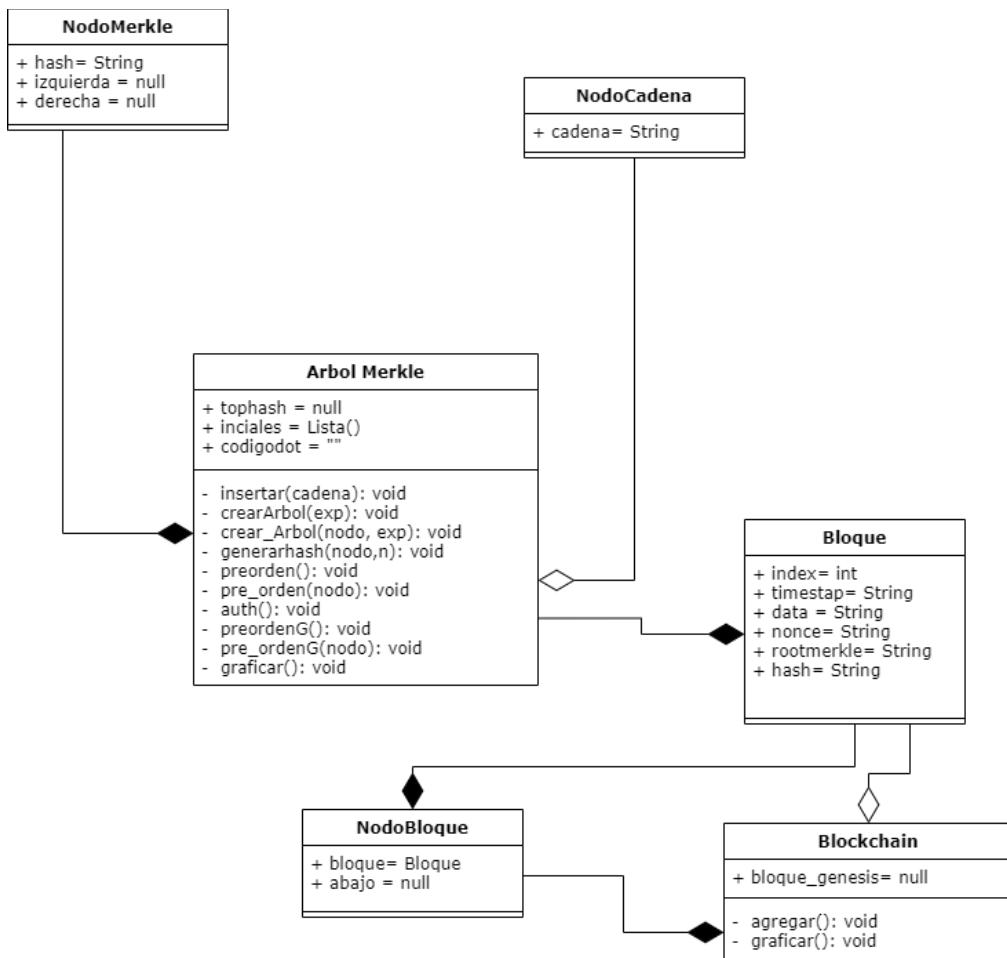
### 4. Tabla Hash



## 5. Árbol de Merkle



## 6. Blockchain



## 7. Diagrama de todo el sistema

