
Universidad Diego Portales
ESCUELA DE INFORMÁTICA &
TELECOMUNICACIONES


# Laboratorio 2: Votación


*Autores:*
Valentina Mella
Valentina Diaz

Profesor:
Marcos Fantoval


9 de abril 2025

# Índice

# 1. Introducción

El presente informe describe de forma detallada el procedimiento realizado para llevar a cabo el laboratorio número dos de la presente materia Estructura de Datos y Algoritmos. A lo largo de este laboratorio, el equipo se enfoca en realizar un código dentro de la aplicación de IntelliJ IDEA, contando con la estricta condición de crear este código bajo el uso e implemento de listas enlazadas (`LinkedList`) , pilas (`Stack`) y colas (`Queue`) , creando dentro del respectivo programa un sistema de votaciones llamado ´Electo´.

Utilizando programación orientada a objetos en Java, se implementaran cuatro clases nombradas de la siguiente manera: {Voto, Candidato, Votante y UrnaElectoral}. Cada clase contará con sus respectivos métodos por separado. Además de la clásica implementación de int main, para corroborar si el código efectivamente funcionará.

Todo esto con el objetivo de aplicar conocimientos sobre estructuras de datos dinámicas en un contexto práctico y simular un proceso electoral dentro de la Facultad de Ingeniería y Ciencias para señalar al futuro presidente del Centro de Alumnos de la Escuela de Informática y Telecomunicaciones.

## 2. Implementación

El codigo que se mostrara se encuentra en el siguente link:
`https://github.com/valeed28/CodLab1/blob/main/Cod_Lab2_EDA`

La clase 'Voto' almacena la información del votante con los sets y gets. La información que se almacena son los atributos:

id que es un entero único para cada voto.

votanteID, el cual da el ID del votante, se utiliza para evitar votos duplicados.

candidatoID es el ID del candidato.

Timestamp es la hora del voto.

```
class Voto {
    private int id;
    private int votanteId;
    private int candidatoId;
    private String timeStamp;

    public Voto(int id, int votanteId, int candidatoId) {
        this.id = id;
        this.votanteId = votanteId;
        this.candidatoId = candidatoId;
        this.timeStamp = LocalTime.now().format(DateTimeFormatter
        .ofPattern("HH:mm:ss")); }

    public int getId() {
     return id;
   }
    public int getVotanteId()
 {
     return votanteId;
 }
    public int getCandidatoId()
{
      return candidatoId;
 }
    public String getTimeStamp() {
      return timeStamp;
}

public void setId(int id) {
this.id = id;
```

```
}
public void setVotanteId(int votanteId) {
this.votanteId = votanteId;
}
    public void setCandidatoId(int candidatoId) {
this.candidatoId = candidatoId;
}
    public void setTimeStamp(String timeStamp) {
this.timeStamp = timeStamp;
   }
}
```

La clase Candidato contiene la información del candidato y gestiona los votos asociados en una cola.

```
class Candidato {
    private int id;
    private String nombre;
    private String partido;
    private Queue<Voto> votosRecibidos;

    public Candidato(int id, String nombre, String partido) {
        this.id = id;
        this.nombre = nombre;
        this.partido = partido;
        this.votosRecibidos = new LinkedList<>();}

    public int getId(){
        return id;
     }
    public String getNombre(){
        return nombre;
}
    public String getPartido() {
        return partido;

    }
    public Queue<Voto> getVotosRecibidos(){
        return votosRecibidos;
    }
    public void setId(int id) {
        this.id = id;
    }
    public void setNombre(String nombre){
```

```
        this.nombre = nombre;
    }
    public void setPartido(String partido) {
        this.partido = partido;
    }
    public void agregarVoto(Voto v) {
        votosRecibidos.offer(v);
        }
    public Voto removerVoto(int idVoto) {
        for (Voto v : votosRecibidos) {
            if (v.getId() == idVoto) {
                votosRecibidos.remove(v);
                return v;
            }
        }
        return null;
    }
}
```

La clase Votante contiene el ID y nombre del votante, junto con un booleano para comprobar si ya ha votado.

```
class Votante {

    private int id;
    private String nombre;
    private boolean yaVoto;

    public Votante(int id, String nombre) {
        this.id = id;
        this.nombre = nombre;
        this.yaVoto = false;
        }
    public int getId() {
        return id;
    }
    public String getNombre() {
        return nombre;
        }
    public boolean getYaVoto() {
        return yaVoto;
        }
    public void setId(int id) {
        this.id = id;
```

```
        }
    public void setNombre(String nombre) {
        this.nombre = nombre;
        }
    public void setYaVoto(boolean yaVoto) {
        this.yaVoto = yaVoto;
        }
    public void marcarComoVotado() {
        this.yaVoto = true;
        }
}
```

La clase UrnaElectoral contiene una lista de candidatos, una pila con el historial de los votos, una cola de los votos anulados o impugnados y un contador para votos.

```
class UrnaElectoral {
    private LinkedList<Candidato> listaCandidatos;
    private Stack<Voto> historialVotos;
    private Queue<Voto> votosReportados;
    private int idCounter;
    private Map<Integer, Votante> votantes;

    public UrnaElectoral() {
        this.listaCandidatos = new LinkedList<>();
        this.historialVotos = new Stack<>();
        this.votosReportados = new LinkedList<>();
        this.idCounter = 1;
        this.votantes = new HashMap<>();}

    public void agregarCandidato(Candidato candidato) {
        listaCandidatos.add(candidato);}

    public void agregarVotante(Votante votante) {
        votantes.put(votante.getId(), votante);}

    public boolean verificarVotante(Votante votante) {
        return !votante.getYaVoto();}

    public boolean registrarVoto(Votante votante, int
       candidatoId) {
        if (!verificarVotante(votante)) {
            System.out.println("El votante " + votante.
               getNombre() + " ya ha votado.");
            return false;}
```

```
        Candidato candidato = listaCandidatos.stream().
            filter(c -> c.getId() == candidatoId).findFirst()
            .orElse(null);

        if (candidato == null) {
            System.out.println("Candidato con ID " +
                candidatoId + "no existe.");
            return false;}

        Voto voto = new Voto(idCounter++, votante.getId(),
            candidatoId);

        candidato.agregarVoto(voto);
        historialVotos.push(voto);
        votante.marcarComoVotado();

        System.out.println("Voto registrado correctamente.
            ID del voto: " + voto.getId());
        return true;}

    public boolean reportarVoto(Candidato candidato, int
        idVoto) {
        Voto voto = candidato.removerVoto(idVoto);

        if (voto == null) {
            System.out.println("Voto con ID " + idVoto + "
                no encontrado en la cola del candidato.");
            return false;}

        votosReportados.add(voto);
        System.out.println("Voto con ID " + idVoto + "
            reportado y movido a la cola de votos reportados.
            ");
        return true;}

    public String obtenerResultados() {
        StringBuilder resultados = new StringBuilder();
        resultados.append("Resultados de la elecci n:");

        for (Candidato candidato : listaCandidatos) {
resultados.append(candidato.getNombre()).append(" (").append
    (candidato.getPartido()).append("): ").append(candidato.
```

```
    getVotosRecibidos().size()).append(" votos");

}

        resultados.append("Total de votos emitidos: ").
            append(historialVotos.size());
        resultados.append("Votos reportados: ").append(
            votosReportados.size());
        return resultados.toString();}
}
```

## 2.1. Clases Auxiliares

Como clase auxiliar se implementó la clase main para mostrar por pantalla una prueba que simula una elección con 2 candidatos y 3 votantes. Se prueba el registro de votos, detección de votos duplicados y reporte de votos

```
public class Main {
    public static void main(String[] args) {

        UrnaElectoral urna = new UrnaElectoral();

        Candidato candidato1 = new Candidato(1, "Juan", "
            Republicano");
        Candidato candidato2 = new Candidato(2, "Isidora", "
            Comunista");
        Candidato candidato3 = new Candidato(3, "Javiera", "
            Socialista");

        urna.agregarCandidato(candidato1);
        urna.agregarCandidato(candidato2);
        urna.agregarCandidato(candidato3);

        Votante votante1 = new Votante(101, "Benjamin");
        Votante votante2 = new Votante(102, "Josefa");
        Votante votante3 = new Votante(103, "Martina");
        Votante votante4 = new Votante(104, "Javier");
        Votante votante5 = new Votante(105, "Sofia");

        urna.agregarVotante(votante1);
        urna.agregarVotante(votante2);
        urna.agregarVotante(votante3);
        urna.agregarVotante(votante4);
        urna.agregarVotante(votante5);
```

```
        urna.registrarVoto(votante1, 1);
        urna.registrarVoto(votante2, 2);
        urna.registrarVoto(votante3, 1);
        urna.registrarVoto(votante4, 3);
        urna.registrarVoto(votante5, 2);
        urna.registrarVoto(votante1, 2);

        if (candidato1.getVotosRecibidos().peek() != null) {
            urna.reportarVoto(candidato1, candidato1.
                getVotosRecibidos().peek().getId());}
        System.out.println("-" + urna.obtenerResultados());}
}
```

## 2.2. Pruebas realizadas

```
Voto registrado correctamente. ID del voto: 1
Voto registrado correctamente. ID del voto: 2
Voto registrado correctamente. ID del voto: 3
Voto registrado correctamente. ID del voto: 4
Voto registrado correctamente. ID del voto: 5
El votante Benjamin ya ha votado.
Voto con ID 1 reportado y movido a la cola de votos reportados.

Resultados de la eleccion:
Juan (Republicano): 1 votos
Isidora (Comunista): 2 votos
Javiera (Socialista): 1 votos

Total de votos emitidos: 5
Votos reportados: 1

=== Code Execution Successful ===
```

Figura 1: Resultado a la hora de ejecutar el codigo

# 3. Análisis de complejidad y uso de memoria

Este código presenta un diseño eficiente para procesos electorales masivos,manteniendo complejidades lineales (O(n)) en sus operaciones más demandantes.

Cuadro 1: Análisis de Complejidad

| **Operación** | **Complejidad** | **Descripción** |
|---|---|---|
| Verificar votante | O(1) | Consulta directa al estado |
| Registrar voto | O(n) | Búsqueda en lista de candidatos |
| Reportar voto | O(m) | Búsqueda en votos del candidato |
| Obtener resultados | O(n) | Conteo por cada candidato |

Al utilizar estructuras como listas enlazadas, pilas y colas que optimizan el uso de memoria (640MB para 10M votos), se garantiza un rendimiento escalable para procesos electorales masivos, manteniendo la integridad y velocidad requeridas en este tipo de aplicaciones críticas.

# 4. Ventajas

Entre las mayores ventajas que se pueden mencionar de haber utilizado listas enlazadas en vez de arreglos en el código, están:

1. **Flexibilidad en tamaño**: Permite que el sistema crezca dinámicamente sin límites predefinidos, evitando la necesidad de redimensionar estructuras como los arreglos, ideal para elecciones con número impredecible de votantes.
2. **Inserción y eliminación eficientes**: Operaciones como agregar votos (pila) o reportarlos (cola) se realizan en tiempo constante O(1), a diferencia de los arreglos donde sería O(n).
3. **Ahorro dinámico de memoria**: Solo se consume memoria por los votos efectivamente emitidos, evitando el desperdicio de espacio que generan los arreglos sobredimensionados.
4. **Manejo óptimo de crecimientos desiguales** : La alta votación de un candidato no impacta el rendimiento del sistema, ya que las estructuras crecen de forma independiente sin necesidad de reubicaciones masivas.
5. **Mejor manejo de concurrencia**: Las listas enlazadas facilitan operaciones seguras en ambientes concurrentes, reduciendo el riesgo de corrupción de datos frente a accesos paralelos.

# 5. Desventajas

Entre las mayores desventajas que se pueden mencionar de haber utilizado listas enlazadas en vez de arreglos en el código, están:

1. **Acceso lento a elementos**: Al ser una lista enlazada, se debe recorrer más veces para llegar al elemento solicitado, mientras que en un arreglo es cosa de seleccionarlo y ya. Esto crearía un problema a la hora de querer buscar o mostrar candidatos o votos específicos; la (`LinkedList`) sería mucho más lenta.
2. **Orden y búsquedas menos eficientes**: Al igual que lo señalado en el primer punto, afecta directamente a la búsqueda y el orden de los elementos, volviéndolos más lentos, mientras que en un arreglo se puede utilizar la búsqueda binaria para facilitar este tipo de tareas, facilitando la notación a un O(log (n)), en vez de un O(n).
3. **Operaciones aleatorias complejas**: En caso de querer implementar funcionalidades más avanzadas como editar un candidato o voto al azar, es muy costoso, ya que nuevamente deberías mover nodo por nodo para realizar este tipo de acciones, mientras que con los arreglos termina siendo más fácil acceder a este tipo de datos y hacer las modificaciones deseadas.

# 6. Conclusiones

Finalmente, se puede determinar que el código realizado fue ejecutado con éxito, logrando demostrar que el equipo logró implementar de manera adecuada y efectiva el uso de listas enlazadas (`LinkedList`) , pilas (`Stack`) y colas (`Queue`) para el sistema de votación ´Electo´. A través de la aplicación IntelliJ IDEA, se logró idear un código con las cuatro clases correspondientes, además de un Int Main auxiliar para validar el correcto funcionamiento del sistema, a través de diversos métodos. Con la aplicación sobre utilización orientada a objetos fue posible idear un código sencillo pero extenso, que lograba catalogar a los votantes por su ID y que a su vez corroboraba si estos ya habían votado. Además, aplicando la clase urna electoral, que a pesar de ser la más complicada de las clases ya mencionadas, se logró una ejecución afinada en concordancia con lo solicitado, agregando una mezcla entre pilas y colas para lograr los objetivos establecidos.

A pesar de los contratiempos encontrados durante la creación de algunas clases, se puede afirmar que la ejecución de este laboratorio fue acertada, alcanzando los objetivos propuestos y demostrandose todo lo anterior a través de la complejidad calculada, la cual se mantuvo siempre lineal O(n).