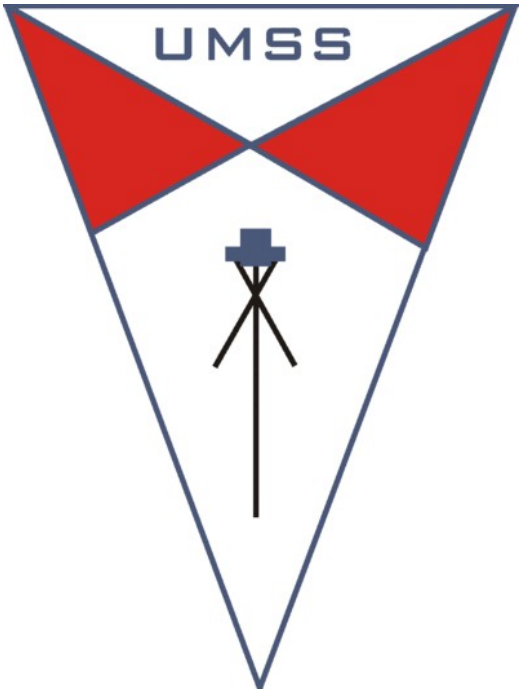
UNIVERSIDAD MAYOR DE “SAN SIMÓN”

FACULTAD DE CIENCIAS Y TECNOLOGÍA



MODELO MATEMÁTICO PARA LAS ELECCIONES GENERALES EN BOLIVIA

Universitario: ISAAC GUTIÉRREZ HUARACHI

Carrera: INGENIERIA DE SISTEMAS

Docente: HENRRY FRANK VILLARROEL TAPIA

Fecha: 21 de mayo de 2025

COCHABAMBA - BOLIVIA

INTRODUCCIÓN

El 17 de agosto de 2025 se llevarán a cabo las elecciones generales de Bolivia para que podamos elegir nuestro presidente(a), vicepresidente(a) y representantes ante la Asamblea Legistativa del Estado Plurinacional de Bolivia. Siendo que es posible que se dé una segunda vuelta el 19 de octubre de 2025.

Como votantes bolivianos se elegirán al presidente(a) y vicepresidente(a) de Bolivia, 130 miembros de la Cámara de Diputados de Bolivia y 36 Integrantes de la Cámara de Senadores de Bolviia para el periodo 2025-2030. El 8 de noviembre de 2025 se realiza la toma posesion de autoridades electas como Presidente y Vicepresidente.

DESCRIPCIÓN DEL PROBLEMA

El análisis que se pretende realizar es el de prever o predecir los votos en base a ciertos parametros como la desviacion estadar o media mediante foirmulas Gaussianas y de MonteCarlo.

SOLUCIÓN PROPUESTA

Lo que se ha propuesto es desarrollar un programa en Java y Excel el cual será capaz de predecir mediante algunos parametros pasados la tendencia en las elecciones del pueblo boliviano y sus votos para el mismo.

Objetivo General

Obtener una predicción de como serán los porcentajes de votos y victorias para las elecciones de Agosto de 2025 o si llegaría a existir segundas rondas.

Objetivo Especifico

- Obtener el Promedio de Votos, Promedio de Escaños al Senado, Promedio de Escaños a Diputados, Probabilidad de Victoria y otros datos a nivel nacional.
- Obtener el Promedio de Votos, Promedio de Escaños al Senado, Promedio de Escaños a Diputados, Probabilidad de Victoria y otros datos a nivel departamental.
- Comparar los resultados obtenidos en el programa de Java con los hechos obtenidos hechos en formulas de hojas en Excel.

IMPLEMENTACIÓN

Modelo en Java

En Java se creo el modelo en base al IDE NetBeans 25 en el cual se usó Java 21, asi que se recomienda la instalación de Java 21 o superiores y Netbeans 21+.

Se usaron 2 librerias que estaran en lo hecho incluidas y estas son jfreechart1.0.19 y jfreechart1.0.23

En Java se crearon 6 clases, divididas en 2 carpetas o packages llamadas model y view.

En model se encuentran las clases:

Clase: `Partido`

La clase `Partido` representa a un partido político dentro de la simulación electoral. Contiene información estadística y visual que se usará en el proceso de simulación basado en el modelo de Monte Carlo.

Atributos:

Atributo	Tipo	Descripción
`nombre`	`String`	Nombre del partido político.
`media`	`double`	Valor medio de la intención de voto estimado para el partido. Se usa en la distribución normal de la simulación.
`desviacion`	`double`	Desviación estándar asociada a la intención de voto del partido. Representa la incertidumbre o variabilidad.
`fotoCandidato`	`File`	Archivo de imagen que representa al candidato del partido.
`logoPartido`	`File`	Archivo de imagen con el logotipo del partido político.
`colorHex`	`String`	Representación en formato hexadecimal del color característico del partido (ej. `#FF0000` para rojo).

`Constructor:`

```
public Partido(String nombre, double media, double desviacion, File
fotoCandidato, File logoPartido, String colorHex)
```

Inicializa un objeto `Partido` con todos los datos relevantes para la simulación.

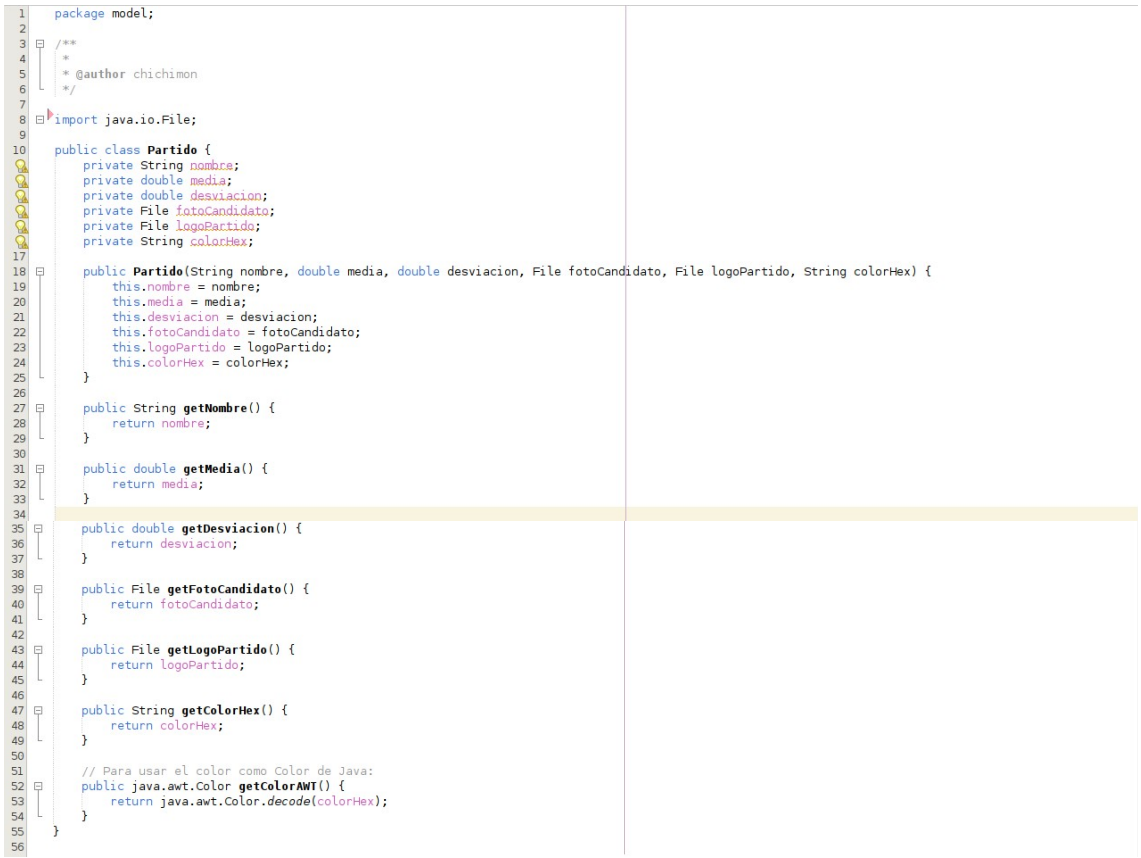
Metodos:

- `getNombre()`: Devuelve el nombre del partido.
- `getMedia()`: Devuelve la media de intención de voto.
- `getDesviacion()`: Devuelve la desviación estándar.
- `getFotoCandidato()`: Devuelve la imagen del candidato.
- `getLogoPartido()`: Devuelve el logo del partido.
- `getColorHex()`: Devuelve el color en formato hexadecimal.

Metodo Adicional

- `getColorAWT()`: Convierte el color hexadecimal (`String`) a un objeto `Color` de la librería AWT de Java para su uso gráfico (por ejemplo, en gráficos de barras o pastel).

Resumen del rol en la simulación:
Esta clase es clave para almacenar y organizar la información de cada partido político. La media y la desviación estándar son parámetros esenciales para simular resultados aleatorios basados en distribuciones normales (modelo de Monte Carlo). Las imágenes y el color permiten generar una interfaz visual más intuitiva y representativa.



```java
package model;

/**
 *
 * @author chichimon
 */

import java.io.File;

public class Partido {
    private String nombre;
    private double media;
    private double desviacion;
    private File fotoCandidato;
    private File logoPartido;
    private String colorHex;

    public Partido(String nombre, double media, double desviacion, File fotoCandidato, File logoPartido, String colorHex) {
        this.nombre = nombre;
        this.media = media;
        this.desviacion = desviacion;
        this.fotoCandidato = fotoCandidato;
        this.logoPartido = logoPartido;
        this.colorHex = colorHex;
    }

    public String getNombre() {
        return nombre;
    }

    public double getMedia() {
        return media;
    }

    public double getDesviacion() {
        return desviacion;
    }

    public File getFotoCandidato() {
        return fotoCandidato;
    }

    public File getLogoPartido() {
        return logoPartido;
    }

    public String getColorHex() {
        return colorHex;
    }

    // Para usar el color como Color de Java:
    public java.awt.Color getColorAWT() {
        return java.awt.Color.decode(colorHex);
    }
}
```

Clase: DHondt

Esta clase implementa el **método D'Hondt**, un sistema matemático de reparto proporcional de escaños utilizado en elecciones parlamentarias. Su función principal es distribuir un número fijo de escaños entre partidos según la cantidad de votos obtenidos.

Método Principal:

```
public static Map<String, Integer> asignarEscanos(Map<String, Double> votos, int escanos)
```

Parámetros:

- `votos`: Un mapa que relaciona el nombre de cada partido (`String`) con la cantidad de votos obtenidos (`Double`).
- `escanos`: El número total de escaños disponibles a repartir (por ejemplo, número de diputados o senadores).

Retorno:

Un `Map<String, Integer>` que indica cuántos escaños se le asignan a cada partido.

Funcionamiento del método:

1. **Inicialización de escaños asignados**:
   Se crea un mapa (`asignados`) donde inicialmente todos los partidos tienen 0 escaños.

2. **Cálculo de cocientes D'Hondt**:
   Para cada partido, se generan `escanos` cocientes dividiendo su total de votos entre los números 1 hasta `escanos`.
   Ejemplo: si un partido tiene 10000 votos y hay 5 escaños, se generan:
   `10000/1, 10000/2, 10000/3, 10000/4, 10000/5`.

3. **Ordenamiento de cocientes**:
   Se ordenan todos los cocientes generados de todos los partidos en orden descendente.

4. **Asignación de escaños**:
   Se eligen los `escanos` cocientes más altos, y a cada partido correspondiente se le incrementa en 1 su contador de escaños en el mapa `asignados`.

`Ejemplo simplificado:`

```
Map<String, Double> votos = Map.of("Partido A", 10000.0, "Partido B",
6000.0);
int escanos = 3;
Map<String, Integer> resultado = DHondt.asignarEscanos(votos,
escanos);
```

Resultado posible:
`{Partido A=2, Partido B=1}`

Aplicación en la simulación:

Este método es clave para simular la **asignación de escaños de manera realista**, basada en votos simulados que se generan usando el modelo de Monte Carlo. Representa una parte del **cómputo final** de la elección.

```java
package model;
/**
 *
 * @author chichimon
 */
import java.util.*;
public class DHondt {

    public static Map<String, Integer> asignarEscanos(Map<String, Double> votos, int escanos) {
        Map<String, Integer> asignados = new HashMap<>();
        for (String partido : votos.keySet()) {
            asignados.put(partido, 0);
        }

        List<Map.Entry<String, Double>> cocientes = new ArrayList<>();

        for (String partido : votos.keySet()) {
            double voto = votos.get(partido);
            for (int i = 1; i <= escanos; i++) {
                cocientes.add(new AbstractMap.SimpleEntry<>(partido, voto / i));
            }
        }

        cocientes.sort((a, b) -> Double.compare(b.getValue(), a.getValue()));

        for (int i = 0; i < escanos; i++) {
            String partido = cocientes.get(i).getKey();
            asignados.put(partido, asignados.get(partido) + 1);
        }

        return asignados;
    }
}
```

Clase: `Simulador`

Esta clase es responsable de **realizar la simulación electoral** utilizando un modelo probabilístico basado en la distribución normal (modelo de Monte Carlo). También incorpora la lógica para verificar si es necesaria una segunda vuelta electoral, según criterios similares a los que se aplican en elecciones presidenciales de algunos países.

Atributos:

Atributo	Tipo	Descripción
`partidos`	`List<Partido>`	Lista de partidos políticos que participan en la elección.
`rand`	`Random`	Generador de números aleatorios, inicializado con una semilla leída desde archivo.

Constructor:

```
public Simulador(List<Partido> partidos)
```

Inicializa el simulador con una lista de partidos. También se configura el generador de números aleatorios `rand` usando una semilla controlada desde el archivo `simulacion_seed.txt`, lo cual permite **reproducir resultados idénticos** si se usa la misma semilla.

Métodos:

`Map<String, Double> simularVotacion()`

Simula una votación asignando un porcentaje estimado a cada partido político.

Lógica del modelo:

1. Se usa `rand.nextGaussian()` para obtener valores aleatorios con distribución normal.
2. Cada partido tiene una **media** y una **desviación estándar**, y se genera un valor aleatorio según estos parámetros.
3. Los valores negativos se descartan (truncados a 0).
4. Finalmente, todos los valores se **normalizan** para que sumen 100% (representando el total del electorado).

`boolean haySegundaVuelta(Map<String, Double> resultados)`

Determina si es necesaria una segunda vuelta electoral con base en los resultados.

Criterios aplicados (basados en Bolivia y otros países):

- Si el candidato más votado obtiene **más del 50%**, gana en primera vuelta.
- Si obtiene al menos **40% y una ventaja de 10 puntos** sobre el segundo, también gana.
- En otros casos, se requiere una segunda vuelta entre los dos primeros.

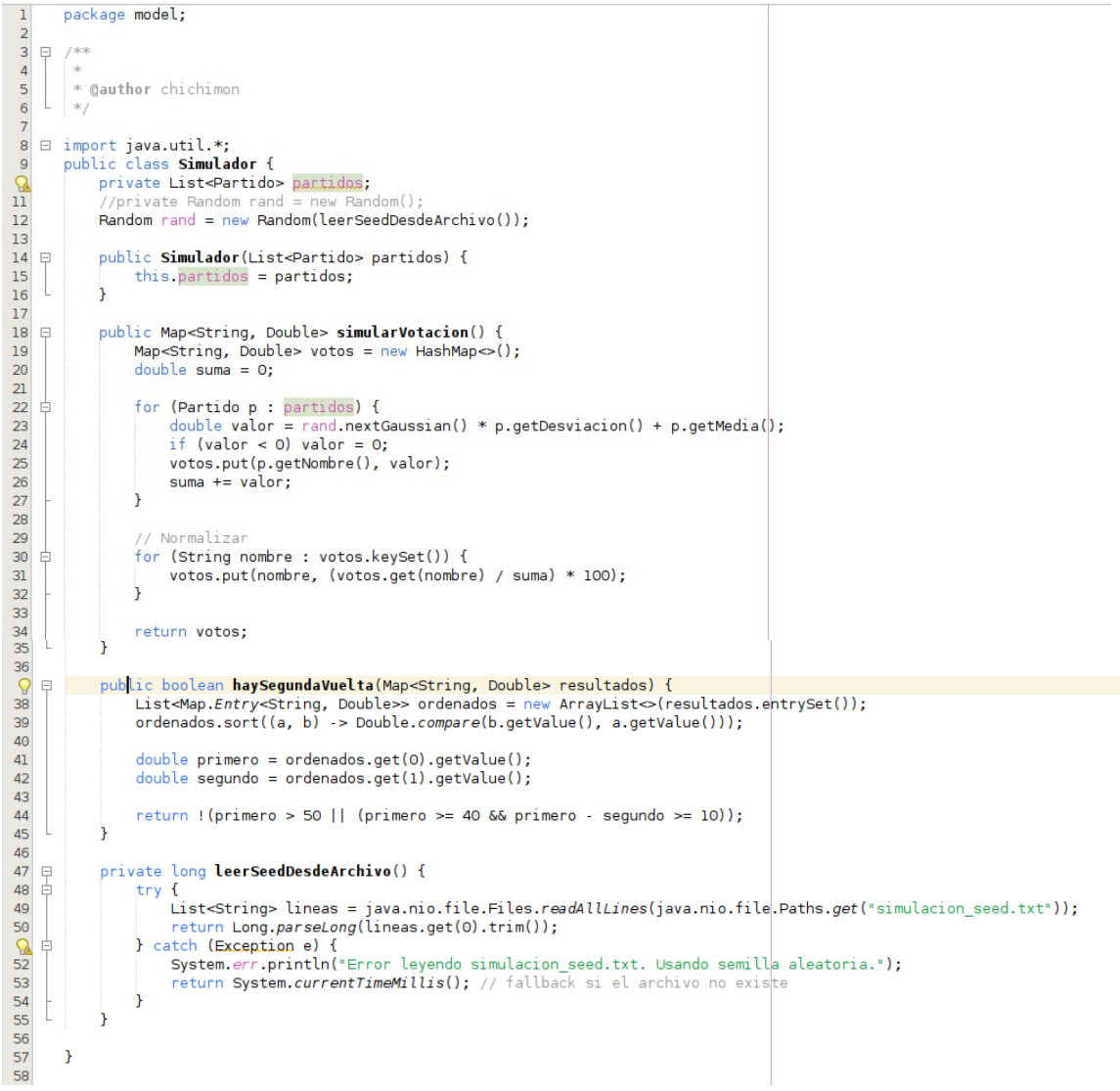
long leerSeedDesdeArchivo()

Lee una semilla (`long`) desde el archivo `simulacion_seed.txt`.

Finalidad:
Permite **controlar la aleatoriedad** de la simulación, útil para pruebas repetibles. Si no se encuentra el archivo o ocurre un error, se usa la hora actual como semilla (resultado aleatorio).

Aplicación en la simulación:

Esta clase es **el núcleo lógico del sistema de simulación electoral**. Genera resultados aleatorios basados en datos estadísticos reales o estimados para cada partido y determina si una elección presidencial requiere segunda vuelta.



```java
package model;

/**
 *
 * @author chichimon
 */

import java.util.*;
public class Simulador {
    private List<Partido> partidos;
    //private Random rand = new Random();
    Random rand = new Random(leerSeedDesdeArchivo());

    public Simulador(List<Partido> partidos) {
        this.partidos = partidos;
    }

    public Map<String, Double> simularVotacion() {
        Map<String, Double> votos = new HashMap<>();
        double suma = 0;

        for (Partido p : partidos) {
            double valor = rand.nextGaussian() * p.getDesviacion() + p.getMedia();
            if (valor < 0) valor = 0;
            votos.put(p.getNombre(), valor);
            suma += valor;
        }

        // Normalizar
        for (String nombre : votos.keySet()) {
            votos.put(nombre, (votos.get(nombre) / suma) * 100);
        }

        return votos;
    }

    public boolean haySegundaVuelta(Map<String, Double> resultados) {
        List<Map.Entry<String, Double>> ordenados = new ArrayList<>(resultados.entrySet());
        ordenados.sort((a, b) -> Double.compare(b.getValue(), a.getValue()));

        double primero = ordenados.get(0).getValue();
        double segundo = ordenados.get(1).getValue();

        return !(primero > 50 || (primero >= 40 && primero - segundo >= 10));
    }

    private long leerSeedDesdeArchivo() {
        try {
            List<String> lineas = java.nio.file.Files.readAllLines(java.nio.file.Paths.get("simulacion_seed.txt"));
            return Long.parseLong(lineas.get(0).trim());
        } catch (Exception e) {
            System.err.println("Error leyendo simulacion_seed.txt. Usando semilla aleatoria.");
            return System.currentTimeMillis(); // fallback si el archivo no existe
        }
    }

}
```

Clase: `MonteCarlo`

Esta clase ejecuta el **modelo de simulación de elecciones basado en el método de Monte Carlo**. Repite la simulación múltiples veces para obtener **estimaciones probabilísticas** sobre:

- Ganadores de presidencia
- Probabilidad de segunda vuelta
- Promedios de porcentaje de votos
- Promedios de escaños en Senado y Diputados (usando el método D'Hondt)

Atributos:

Atributo	Tipo	Descripción
`simulador`	`Simulador`	Objeto que genera simulaciones individuales de votación.
`repeticiones`	`int`	Número de veces que se ejecutará la simulación para obtener estadísticas confiables.

`Constructor:`

```
public MonteCarlo(Simulador simulador, int repeticiones)
```

Recibe una instancia de `Simulador` y el número de repeticiones. Esto permite ejecutar muchas simulaciones consecutivas, lo cual es el núcleo del método de Monte Carlo.

Método Principal:

`void ejecutar()`

Este método realiza las simulaciones y **presenta estadísticas consolidadas**.

📋 **Pasos principales de ejecución:**

1. **Inicialización de variables auxiliares**:
    - `sumaPorcentajes`: acumula porcentajes de votos para calcular promedios.
    - `ganaPresidencia`: cuenta cuántas veces gana cada partido.
    - `segundaVueltaCount`: cuántas veces se requiere segunda vuelta.
    - `totalEscSenado` y `totalEscDip`: total de escaños acumulados por partido en Senado y Diputados.
2. **Simulación repetida** (`for` con `repeticiones`):
    - Se simula una votación.
    - Se determina al ganador de esa simulación.
    - Se acumulan porcentajes de votos por partido.
    - Se verifica si habría segunda vuelta.
    - Se asignan escaños al Senado (4) y Diputados (10) con el método de D'Hondt.
    - Se acumulan los resultados.
3. **Impresión de resultados finales**:

- Probabilidad de segunda vuelta.
- Promedios de voto por partido.
- Frecuencia de victorias presidenciales por partido.
- Promedios de escaños asignados (Senado y Diputados).

Importancia y Aplicación:

La clase `MonteCarlo` representa el **motor de análisis estadístico del sistema electoral simulado**. Permite realizar un análisis **probabilístico y cuantitativo**, lo que ayuda a:

- Estimar con mayor confianza los posibles resultados electorales.
- Observar tendencias a lo largo de múltiples simulaciones.
- Evaluar la competitividad de cada partido en diferentes niveles (presidencia, senado, diputados).
- Mostrar con datos si habría necesidad de segunda vuelta con frecuencia.

```java
package model;

/**
 *
 * @author chichimon
 */

import java.util.*;

public class MonteCarlo {
    private Simulador simulador;
    private int repeticiones;

    public MonteCarlo(Simulador simulador, int repeticiones) {
        this.simulador = simulador;
        this.repeticiones = repeticiones;
    }

    public void ejecutar() {
        Map<String, Double> sumaPorcentajes = new HashMap<>();
        Map<String, Integer> ganaPresidencia = new HashMap<>();
        int segundaVueltaCount = 0;

        Map<String, Integer> totalEscSenado = new HashMap<>();
        Map<String, Integer> totalEscDip = new HashMap<>();

        for (int i = 0; i < repeticiones; i++) {
            Map<String, Double> votos = simulador.simularVotacion();
            List<Map.Entry<String, Double>> orden = new ArrayList<>(votos.entrySet());
            orden.sort((a, b) -> Double.compare(b.getValue(), a.getValue()));
            String ganador = orden.get(0).getKey();

            ganaPresidencia.put(ganador, ganaPresidencia.getOrDefault(ganador, 0) + 1);

```

```
            for (String partido : votos.keySet()) {
                sumaPorcentajes.put(partido, sumaPorcentajes.getOrDefault(partido, 0.0) + votos.get(partido));
            }

            if (simulador.haySegundaVuelta(votos)) {
                segundaVueltaCount++;
            }

            // Asignación de escaños
            Map<String, Integer> escSenado = DHondt.asignarEscanos(votos, 4);
            for (String p : escSenado.keySet()) {
                totalEscSenado.put(p, totalEscSenado.getOrDefault(p, 0) + escSenado.get(p));
            }

            Map<String, Integer> escDip = DHondt.asignarEscanos(votos, 10);
            for (String p : escDip.keySet()) {
                totalEscDip.put(p, totalEscDip.getOrDefault(p, 0) + escDip.get(p));
            }
        }

        System.out.println("\n===== RESULTADOS DE " + repeticiones + " SIMULACIONES =====\n");

        System.out.println("Probabilidad de segunda vuelta: " + ((segundaVueltaCount * 100.0) / repeticiones) + "%\n");

        System.out.println("--- Promedio de porcentaje de votos ---");
        for (String p : sumaPorcentajes.keySet()) {
            double promedio = sumaPorcentajes.get(p) / repeticiones;
            System.out.printf("%s: %.2f%%\n", p, promedio);
        }

        System.out.println("\n--- Frecuencia de victoria presidencial ---");
        for (String p : ganaPresidencia.keySet()) {
            double porcentaje = ganaPresidencia.get(p) * 100.0 / repeticiones;
```

```
        System.out.println("--- Promedio de porcentaje de votos ---");
        for (String p : sumaPorcentajes.keySet()) {
            double promedio = sumaPorcentajes.get(p) / repeticiones;
            System.out.printf("%s: %.2f%%\n", p, promedio);
        }

        System.out.println("\n--- Frecuencia de victoria presidencial ---");
        for (String p : ganaPresidencia.keySet()) {
            double porcentaje = ganaPresidencia.get(p) * 100.0 / repeticiones;
            System.out.printf("%s: %.2f%%\n", p, porcentaje);
        }

        System.out.println("\n--- Promedio de escaños en Senado ---");
        for (String p : totalEscSenado.keySet()) {
            double promedio = totalEscSenado.get(p) / (double) repeticiones;
            System.out.printf("%s: %.2f\n", p, promedio);
        }

        System.out.println("\n--- Promedio de escaños en Diputados ---");
        for (String p : totalEscDip.keySet()) {
            double promedio = totalEscDip.get(p) / (double) repeticiones;
            System.out.printf("%s: %.2f\n", p, promedio);
        }
    }
}
```

Clase: `ExportadorResultados`

La clase `ExportadorResultados` se encarga de **exportar los resultados de las simulaciones electorales a un archivo .CSV**, además de mostrar una tabla de resumen en consola.

Es una herramienta clave para **guardar y visualizar resultados estadísticos** del modelo de Monte Carlo, útil tanto para análisis como para respaldar evidencias en un proyecto de investigación o tesis.

Método Principal:

```
java
CopiarEditar
public static void exportarCSV(String nombreArchivo,
                               Map<String, Double> porcentajes,
                               Map<String, Integer> victorias,
                               int totalSimulaciones,
                               Map<String, Integer> escSenado,
                               Map<String, Integer> escDiputados,
                               int cuentaSegundaVuelta)
```

Parametros:

Parámetro	Tipo	Descripción
`nombreArchivo`	`String`	Nombre del archivo `.csv` de salida.
`porcentajes`	`Map<String, Double>`	Acumulación de porcentajes de votos por partido.
`victorias`	`Map<String, Integer>`	Número de victorias presidenciales por partido.
`totalSimulaciones`	`int`	Total de simulaciones ejecutadas.
`escSenado`	`Map<String, Integer>`	Total de escaños asignados al Senado por partido.
`escDiputados`	`Map<String, Integer>`	Total de escaños asignados a Diputados por partido.
`cuentaSegundaVuelta`	`int`	Número de simulaciones en las que hubo segunda vuelta.

Funcionalidad detallada:

1. **Creación de carpeta `resultados/`**
    - Verifica si existe; si no, la crea.
    - Guarda los archivos CSV dentro de esta carpeta para mantener organizados los datos.
2. **Escritura del archivo CSV**
    - Usa punto y coma (`;`) como delimitador para compatibilidad con Excel.
    - Cada fila contiene:
      `Partido; Promedio Voto %; Promedio Escaños Senado; Promedio Escaños Diputados; Prob. Victoria Presidencial %; Prob. Segunda Vuelta %`
3. **Visualización en consola**
    - Muestra una tabla alineada con:
        - Voto promedio
        - Escaños promedio (Senado y Diputados)
        - Porcentaje de victoria presidencial
    - Además, muestra la probabilidad de segunda vuelta general.
4. **Manejo de errores**

- Captura excepciones de entrada/salida (IOException).
- Muestra errores en consola si ocurre algún fallo al crear o escribir el archivo.

Ejemplo de salida en consola:

```
✅ Archivo creado: simulacion_1.csv

--- RESULTADOS: SIMULACION_1.CSV ---
Partido    Voto (%)       Escaños Senado       Escaños Diputados
Victoria (%)
MAS        42.10          2.45                 5.30
60.00
CC         27.30          1.15                 2.70
25.00
CREEMOS    20.60          0.30                 1.10
10.00
...

Probabilidad de segunda vuelta: 40.00%
-----------------------------------------------
```

Importancia en el proyecto:

- Permite **guardar evidencia persistente** de cada lote de simulaciones.
- Compatible con herramientas como **Microsoft Excel**, **Google Sheets** o **SPSS** para análisis posteriores.
- Mejora la **usabilidad del sistema** al permitir a los usuarios exportar y revisar resultados fuera del entorno de ejecución.

```java
package model;

/**
 *
 * @author chichimon
 */

import java.io.FileWriter;
import java.io.IOException;
import java.util.Map;
import java.io.File;

public class ExportadorResultados {
    public static void exportarCSV(String nombreArchivo, Map<String, Double> porcentajes,
                                   Map<String, Integer> victorias, int totalSimulaciones,
                                   Map<String, Integer> escSenado, Map<String, Integer> escDiputados,
                                   int cuentaSegundaVuelta) {

        try {
            // Verifica o crea la carpeta 'resultados'
            File carpeta = new File("resultados");
            if (!carpeta.exists()) {
                carpeta.mkdirs();  // crea la carpeta si no existe
            }

            FileWriter writer = new FileWriter("resultados/" + nombreArchivo);
            writer.write("Partido;Promedio Voto %;Promedio Escaños Senado;Promedio Escaños Diputados"
                    + ";Prob Victoria Presidencial %;Prob. Segunda Vuelta %\n");

            for (String partido : porcentajes.keySet()) {
                double promedioVoto = porcentajes.get(partido) / totalSimulaciones;
                double promedioSenado = escSenado.getOrDefault(partido, 0) / (double) totalSimulaciones;
                double promedioDip = escDiputados.getOrDefault(partido, 0) / (double) totalSimulaciones;
                double porcentajeVictoria = victorias.getOrDefault(partido, 0) * 100.0 / totalSimulaciones;
                double probSegundaVuelta = cuentaSegundaVuelta * 100.0 / totalSimulaciones;

                writer.write(String.format("%s;%.2f;%.2f;%.2f;%.2f;%.2f\n",
                        partido, promedioVoto, promedioSenado, promedioDip, porcentajeVictoria, probSegundaVuelta));
            }
            writer.close();
            System.out.println("✔ Archivo creado: " + nombreArchivo);

            // Mostrar en consola (tabla rápida)
            System.out.println("\n--- RESULTADOS: " + nombreArchivo.toUpperCase() + " ---");
            System.out.printf("%-10s %-15s %-20s %-22s %-20s\n",
                "Partido", "Voto (%)", "Escaños Senado", "Escaños Diputados", "Victoria (%)");

            for (String partido : porcentajes.keySet()) {
                double promedioVoto = porcentajes.get(partido) / totalSimulaciones;
                double promedioSenado = escSenado.getOrDefault(partido, 0) / (double) totalSimulaciones;
                double promedioDip = escDiputados.getOrDefault(partido, 0) / (double) totalSimulaciones;
                double porcentajeVictoria = victorias.getOrDefault(partido, 0) * 100.0 / totalSimulaciones;

                System.out.printf("%-10s %-15.2f %-20.2f %-22.2f %-20.2f\n",
                        partido, promedioVoto, promedioSenado, promedioDip, porcentajeVictoria);
            }

            // Mostrar probabilidad de segunda vuelta (una vez por archivo)
            double probSegundaVuelta = cuentaSegundaVuelta * 100.0 / totalSimulaciones;
            System.out.println("Probabilidad de segunda vuelta: " + String.format("%.2f%%", probSegundaVuelta));
            System.out.println("-----------------------------------------------");
        } catch (IOException e) {
            System.err.println("Error al escribir archivo " + nombreArchivo);
        }
    }
}
```

En view, la parte visual del sistema fue diseñada utilizando la biblioteca **Java Swing**, y representa uno de los componentes más complejos del proyecto, tanto en términos de implementación como de interacción con los modelos de simulación. En esta unicamente se encuentra la clase:

Clase Principal: `SimulacionElectoralGUI`

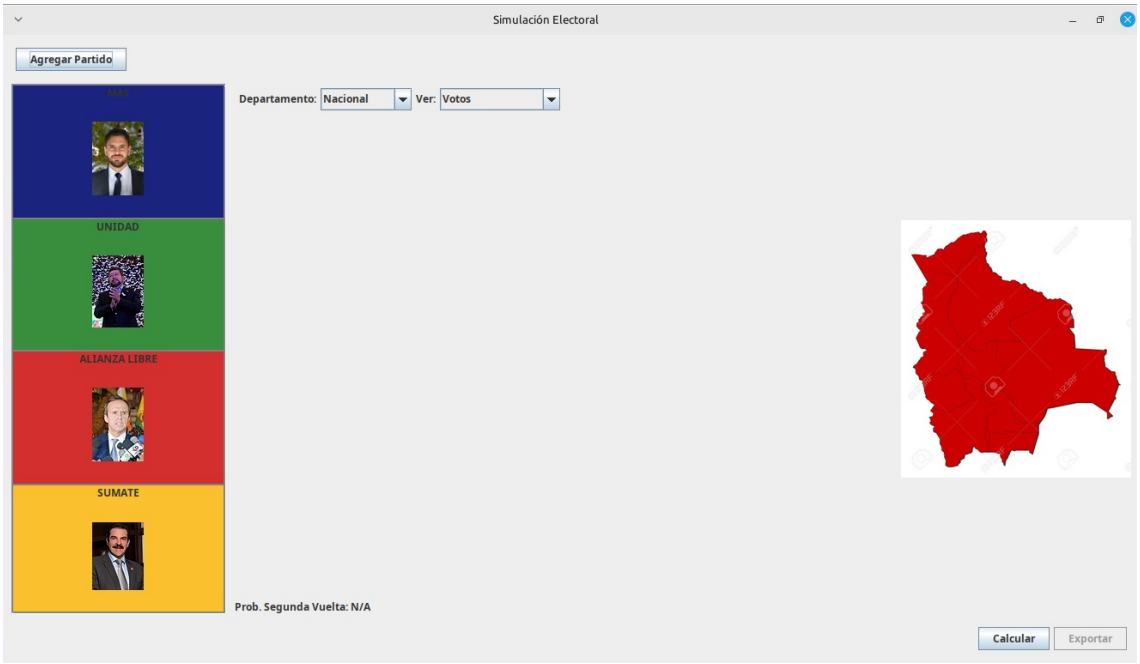
Esta clase extiende `JFrame` y actúa como la **ventana principal del sistema**. Aquí se integran todos los elementos visuales y funcionales para gestionar los partidos, realizar la simulación y visualizar los resultados.

1. **Estructura General de la Ventana**

La interfaz se divide en varias secciones principales utilizando `BorderLayout`:

- **Norte (NORTH)**: Botón para agregar partidos.

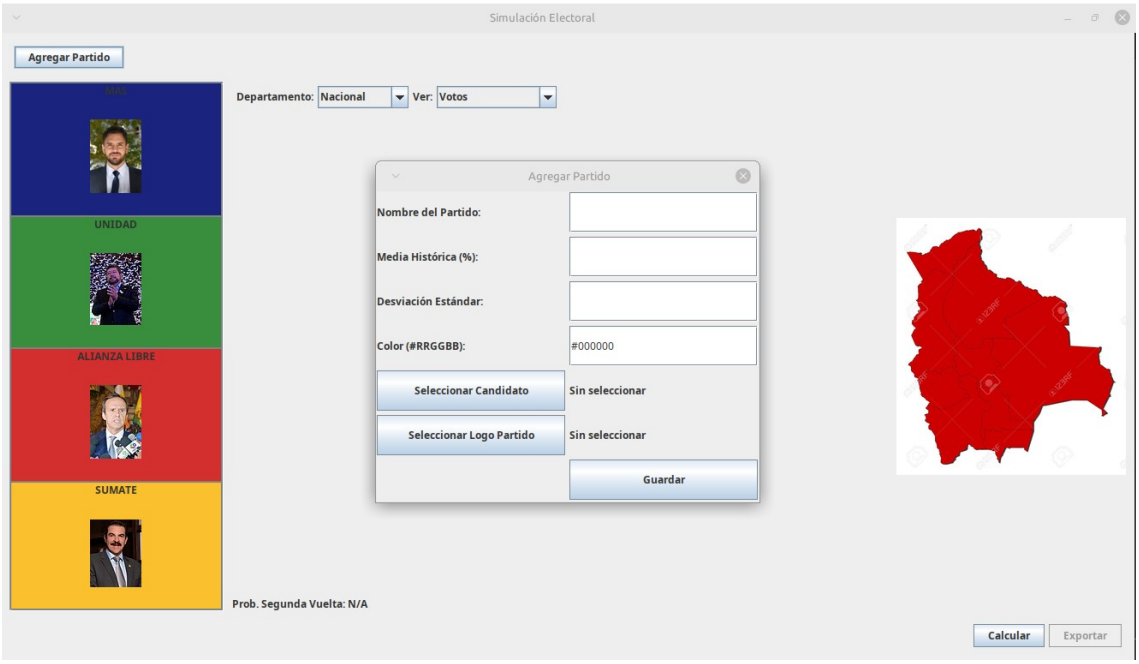
- **Oeste (WEST)**: Panel de desplazamiento que muestra las tarjetas de los partidos agregados.
- **Centro (CENTER)**: Panel central donde se visualizan gráficos de resultados y controles.
- **Este (EAST)**: Muestra el mapa correspondiente al departamento seleccionado.
- **Sur (SOUTH)**: Botones de acciones principales: *Calcular* y *Exportar*.



2. **Agregado Dinámico de Partidos**

Al presionar el botón **"Agregar Partido"**, se abre una ventana (`JDialog`) con los siguientes campos:

- Nombre del partido.
- Media histórica y desviación estándar.
- Color representativo en formato hexadecimal.
- Imagen del candidato y logo del partido (opcional, mediante `JFileChooser`).



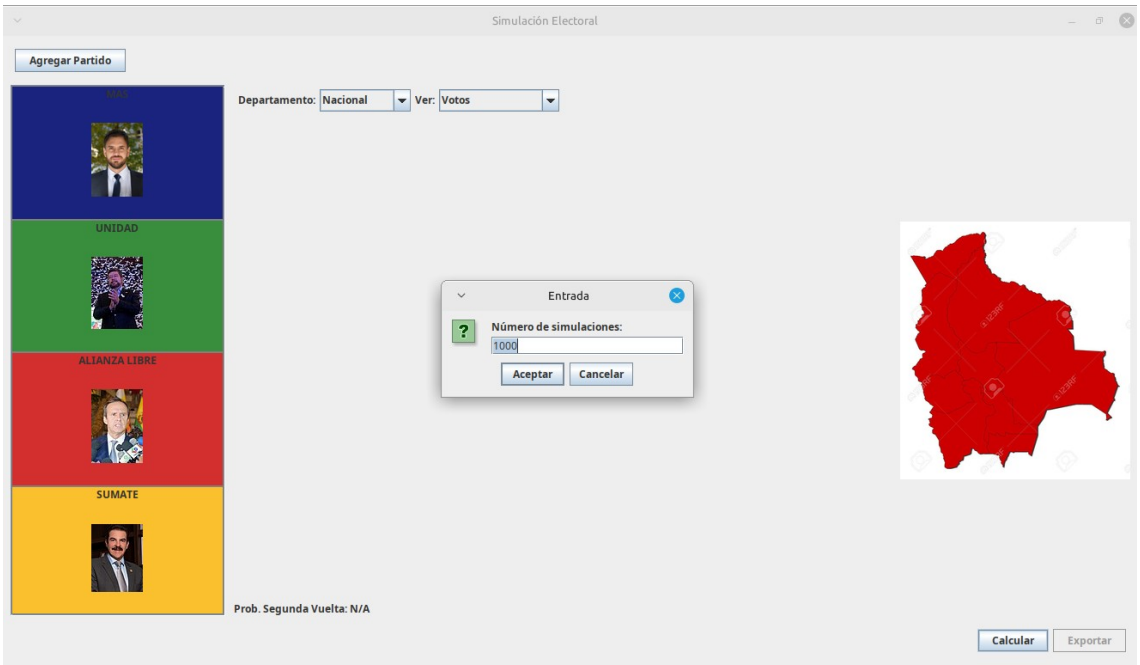
Cada partido se agrega a una lista (`List<Partido>`) y se representa visualmente con una tarjeta (`JPanel`) que incluye su nombre, color y foto, en el panel lateral izquierdo.

Por defecto vienen 4 candidatos, los que se pueden ver pero se pueden agregar mas.

3. **Simulación de Resultados**

Al pulsar **"Calcular"**, se realiza una simulación de Monte Carlo para cada departamento:

- Se ejecutan múltiples iteraciones (definidas por el usuario).



- Se generan votaciones aleatorias por partido usando distribución normal (`nextGaussian`), con parámetros definidos por el usuario.

- Se calcula:
    - **Distribución porcentual de votos.**
    - **Asignación de escaños** mediante el método **D'Hondt** (Senadores y Diputados).
    - **Probabilidad de segunda vuelta** en base al sistema boliviano.
    - **Frecuencia de victoria** de cada partido.
- Los resultados se almacenan en un `Map` jerárquico para poder ser consultados dinámicamente según el departamento seleccionado.

4. **Visualización de Resultados**

Los resultados se pueden visualizar desde un `JComboBox` que permite seleccionar entre:

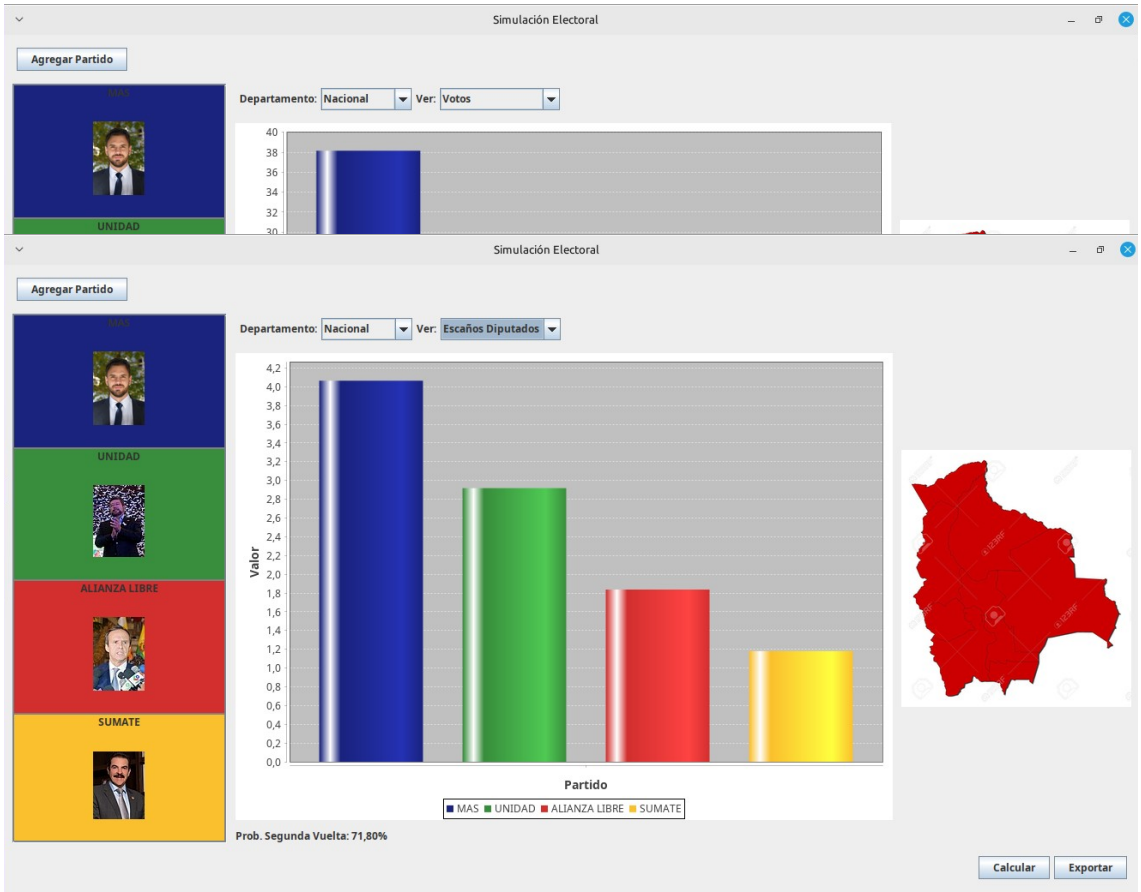
- Votos
- Escaños Senado
- Escaños Diputados
- Victoria

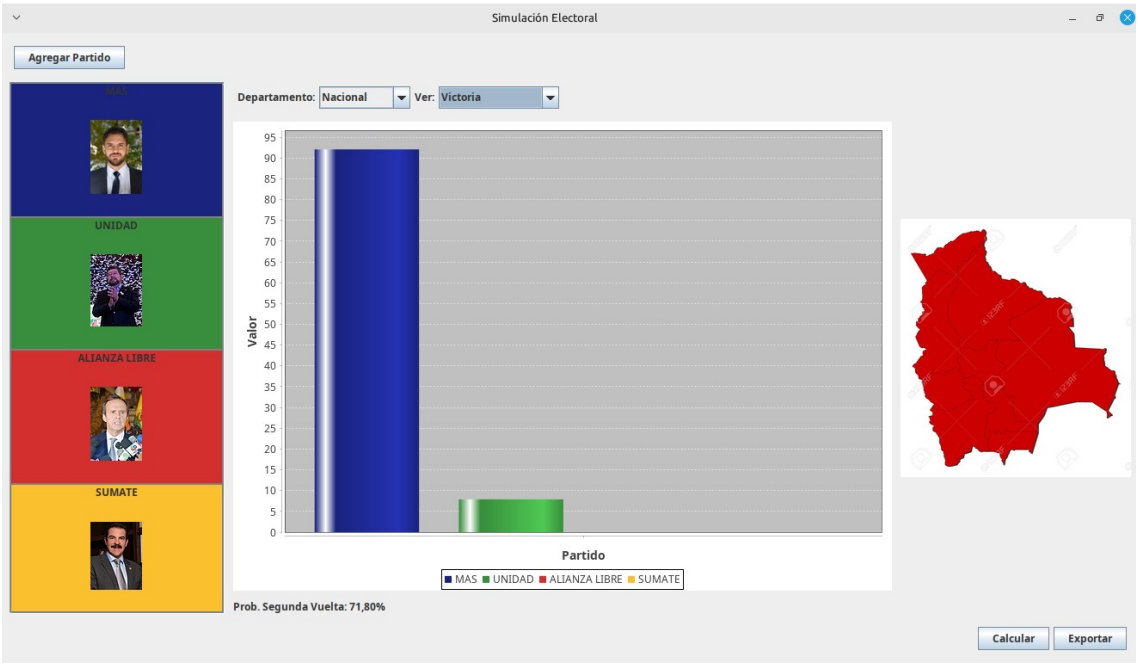
Cada tipo de dato se representa mediante un **gráfico de barras** generado con la biblioteca `JFreeChart`, el cual es agregado dinámicamente al panel `panelChart`.

Ejemplo de flujo:

```
cbChartType.addActionListener(e -> redrawChart());
```

- Se redibuja el gráfico automáticamente al cambiar la métrica a visualizar.
- Los datos utilizados para construir los gráficos provienen del `Map<String, Map<String, Map<String, Double>>> resultadosPorDepto`.

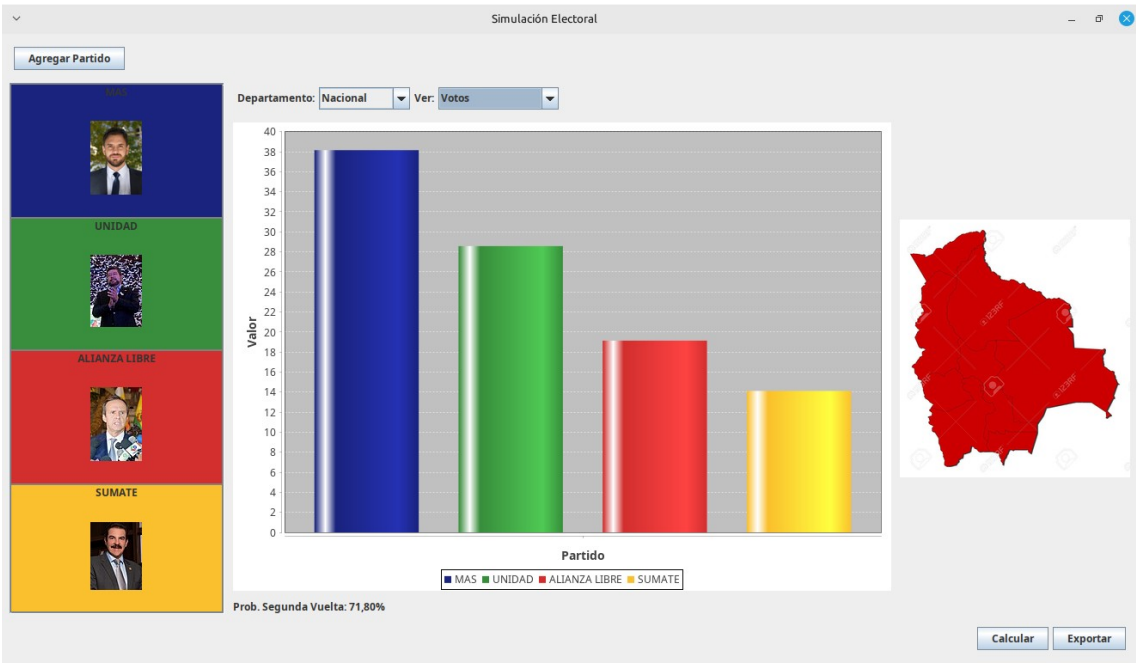




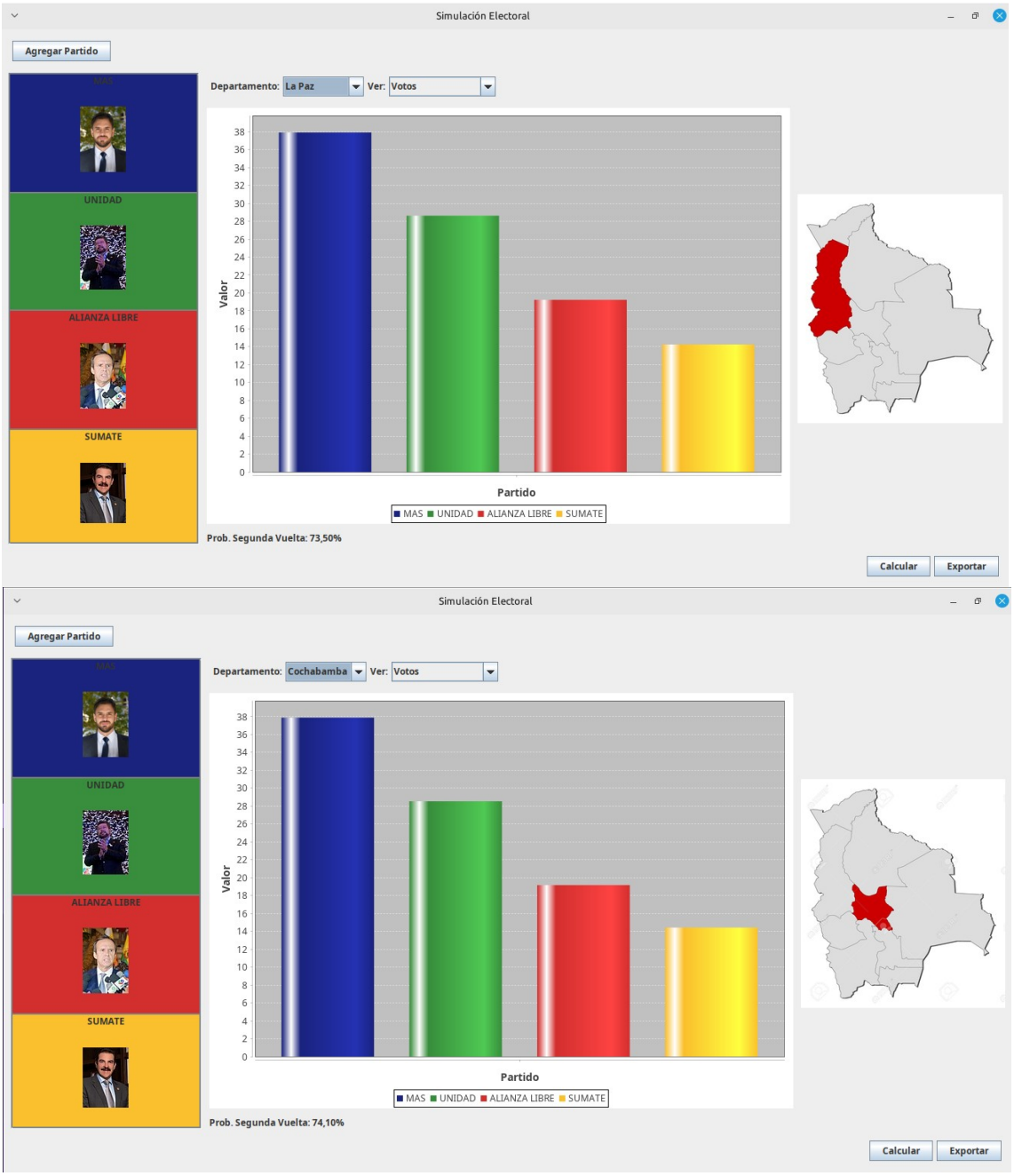
5. **Visualización del Mapa Departamental**

Se cargan imágenes representativas de cada departamento desde archivos (`ImageIcon`) y se muestran en el panel derecho (`lblMap`) de acuerdo al departamento seleccionado, ademas de cambiar ligeramente los resultados dependiendo de la región, pues los resultados no solo abarcan a nivel nacional sino también abarcan de departamento a departamento.

Ejemplo a nivel nacional (por defecto):



Ejemplo a nivel departamental:

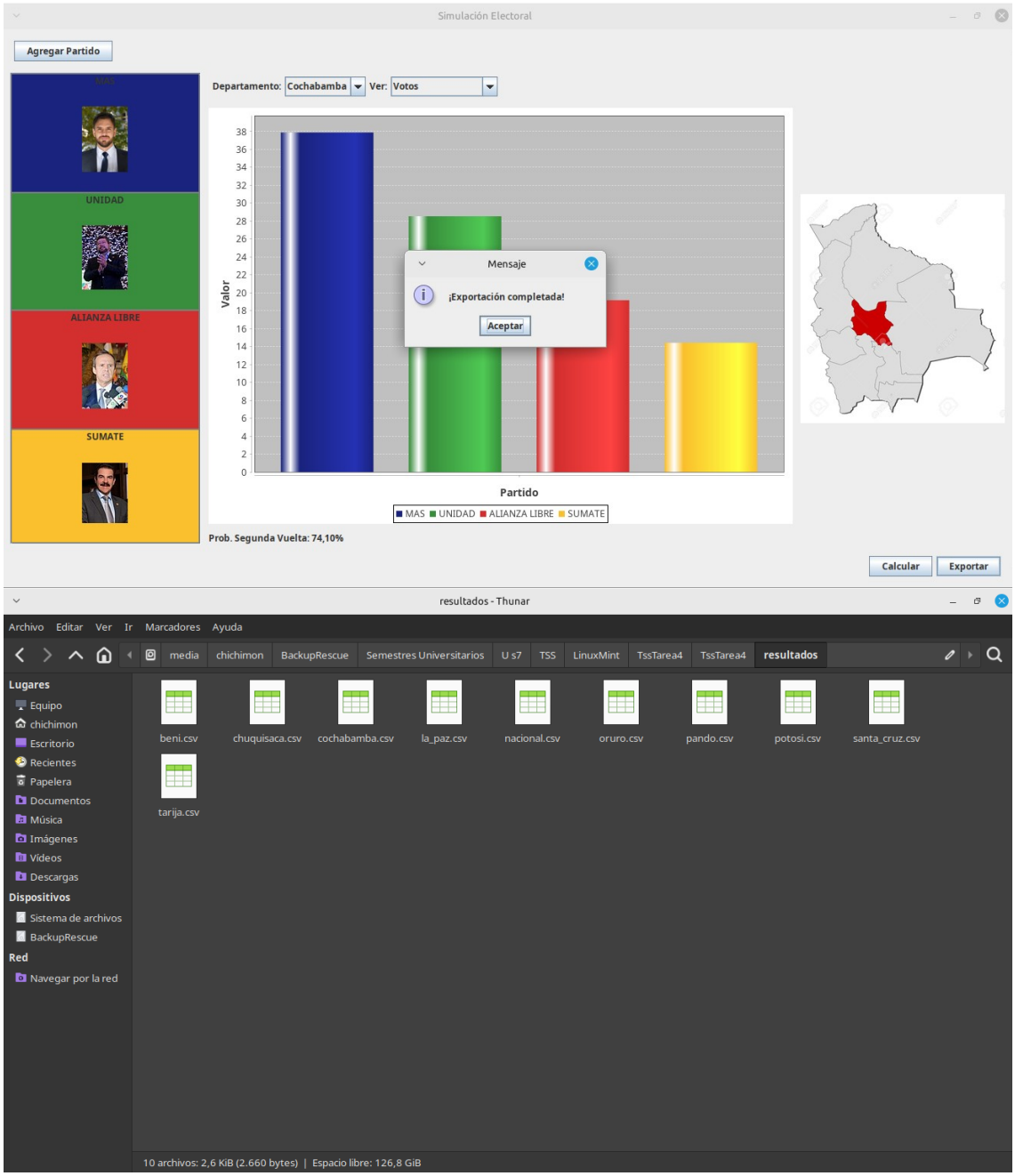


Y asi para los otros 7 departamentos restantes.

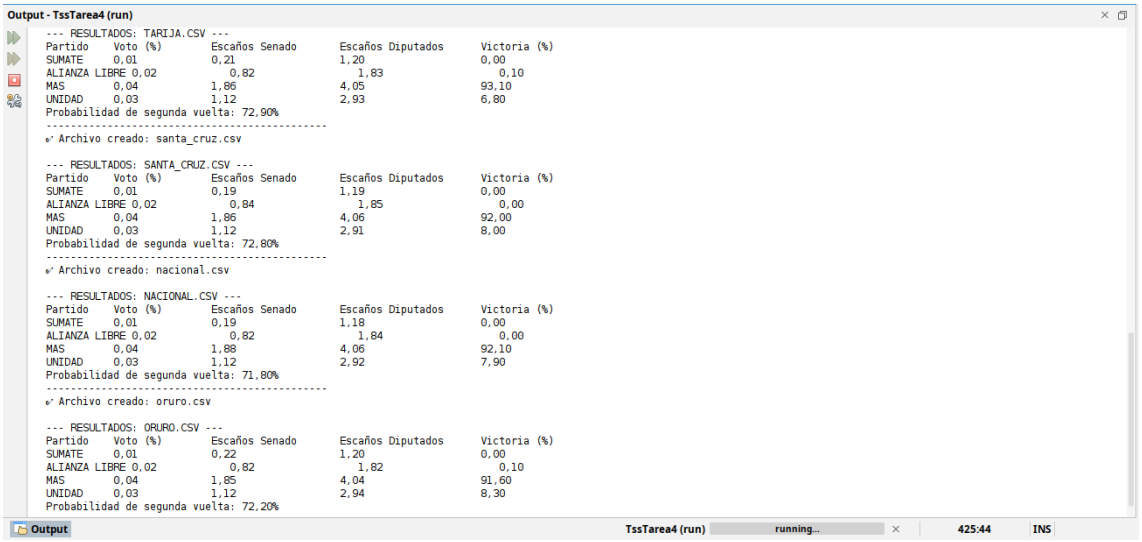
6. **Exportación de Resultados**

Luego de una simulación, el botón **"Exportar"** se habilita y permite guardar los resultados en un archivo Excel. Esta función se encuentra delegada a la clase `ExportadorResultados`.

A



También se pueden ver detallados en la consola de Java:



7. **Otros Aspectos Técnicos Importantes**

- Uso de `BoxLayout` para apilar verticalmente las tarjetas de partidos.
- Uso de `GridLayout` para el diálogo de agregación de partidos.
- Validación de formato de color hexadecimal.
- Lectura de imágenes con `JFileChooser` y `ImageIcon`.
- Refrescado dinámico de componentes mediante `revalidate()` y `repaint()`.

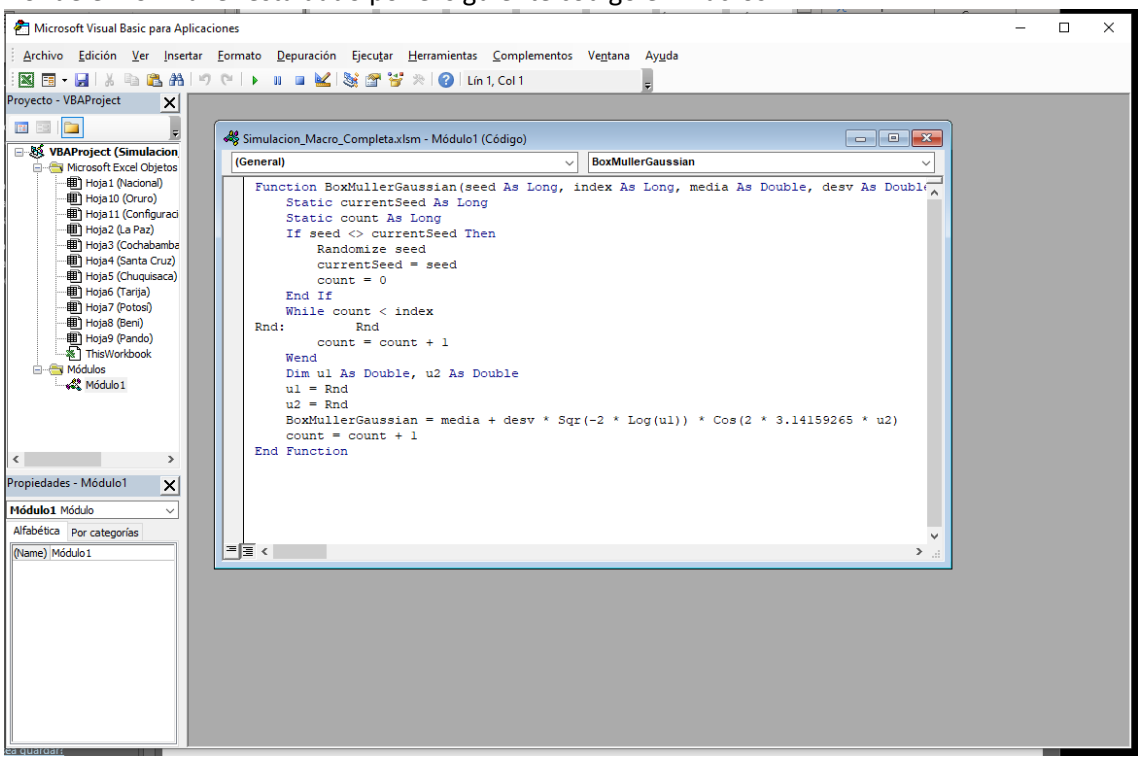
Conclusión:

La clase `SimulacionElectoralGUI` representa el corazón visual del sistema, permitiendo que el usuario interactúe de manera intuitiva con los modelos de simulación. Su diseño modular, junto con el uso de Swing y JFreeChart, proporciona una experiencia gráfica rica y funcional. A través de la integración de múltiples componentes, el sistema facilita tanto la creación de escenarios personalizados como la exploración visual de los resultados simulados.

Modelo en Excel

En Excel lo que hacemos es usando las macros de VBA, posibles de usar en Excel, ponemos los datos de cada partido, en cada hoja calcularemos para cada departamento o a nivel nacional y en una hoja estara la configuracion que hace funcionar el VBA y este ira calculando mediante la formula:
=@BoxMullerGaussian(Configuracion$B$1;NroDeSimulacion;Media;Desviacion Estandar)

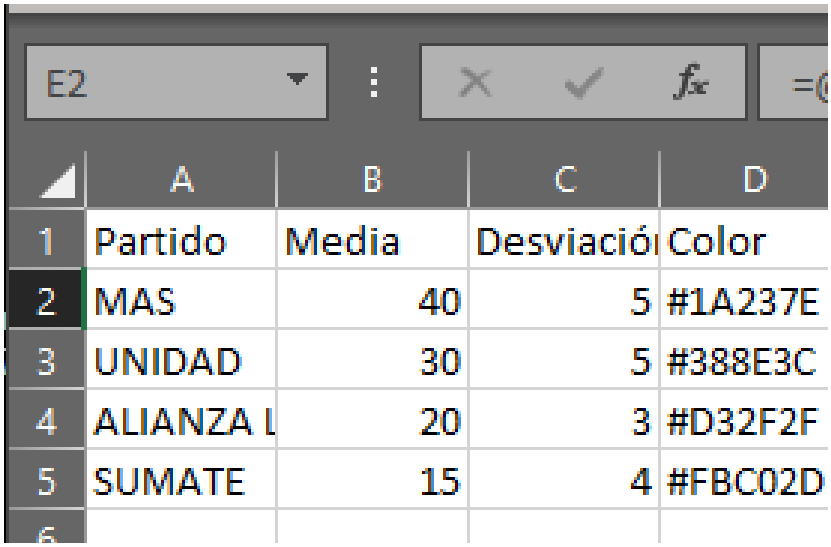
Donde el BoxMuller esta dado por el siguiente codigo en macros:



Este codigo solo es capaz de identificar el porcentaje de votos por ahora.

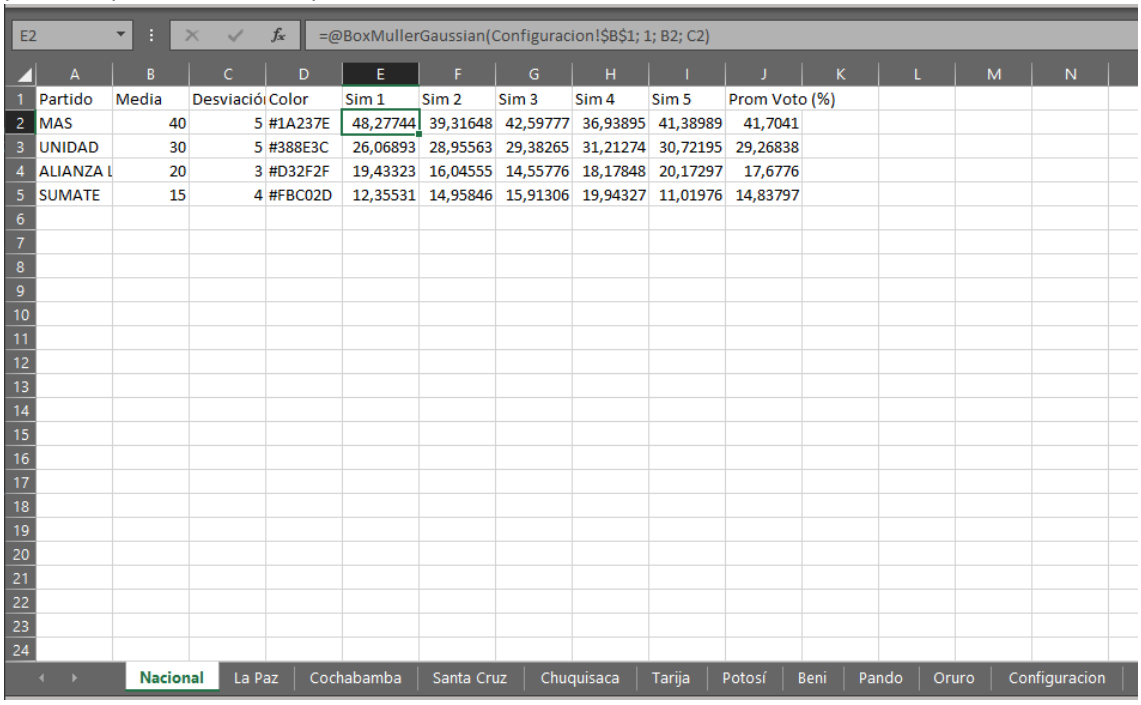
Comparación

Para esto usaremos estos datos como referencia los datos:



Y al compararlos salen muchas diferencias, esto ocurre porque a la hora de hacer la simulacion usan un gaussiano random, el cual no es lo mismo para ambos haciendo que tenga diferencias notorias por este motivo, y por alguna razon se muestra con un decimal cero mas lo cual hace

parecer que son distintos pero en realidad son los mismo resultados en diferentes escalas:



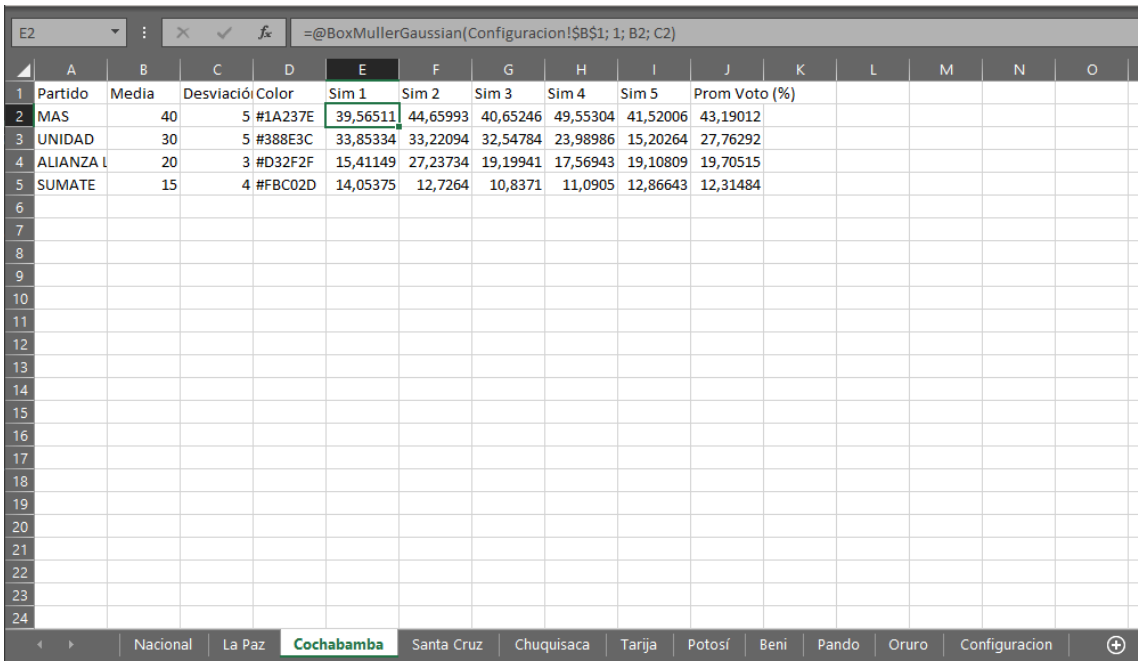
Partido	Media	Desviació	Color	Sim 1	Sim 2	Sim 3	Sim 4	Sim 5	Prom Voto (%)
MAS	40	5	#1A237E	48,27744	39,31648	42,59777	36,93895	41,38989	41,7041
UNIDAD	30	5	#388E3C	26,06893	28,95563	29,38265	31,21274	30,72195	29,26838
ALIANZA L	20	3	#D32F2F	19,43323	16,04555	14,55776	18,17848	20,17297	17,6776
SUMATE	15	4	#FBC02D	12,35531	14,95846	15,91306	19,94327	11,01976	14,83797

Nacional | La Paz | Cochabamba | Santa Cruz | Chuquisaca | Tarija | Potosí | Beni | Pando | Oruro | Configuracion

```
------------------------------------------------
✓ Archivo creado: nacional.csv

--- RESULTADOS: NACIONAL.CSV ---
Partido    Voto (%)        Escaños Senado       Escaños Diputados      Victoria (%)
SUMATE     0,01            0,22                 1,21                   0,10
ALIANZA LIBRE 0,02            0,82                 1,82                   0,00
MAS        0,04            1,88                 4,08                   93,60
UNIDAD     0,03            1,09                 2,89                   6,30
Probabilidad de segunda vuelta: 73,50%
------------------------------------------------
```

A nivel departamental, ejemplo solo para Cbba:



Partido	Media	Desviació	Color	Sim 1	Sim 2	Sim 3	Sim 4	Sim 5	Prom Voto (%)
MAS	40	5	#1A237E	39,56511	44,65993	40,65246	49,55304	41,52006	43,19012
UNIDAD	30	5	#388E3C	33,85334	33,22094	32,54784	23,98986	15,20264	27,76292
ALIANZA L	20	3	#D32F2F	15,41149	27,23734	19,19941	17,56943	19,10809	19,70515
SUMATE	15	4	#FBC02D	14,05375	12,7264	10,8371	11,0905	12,86643	12,31484

Nacional | La Paz | Cochabamba | Santa Cruz | Chuquisaca | Tarija | Potosí | Beni | Pando | Oruro | Configuracion

```
--------------------------------------------
✅ Archivo creado: cochabamba.csv

--- RESULTADOS: COCHABAMBA.CSV ---
Partido    Voto (%)        Escaños Senado       Escaños Diputados      Victoria (%)
SUMATE     0,01            0,19                 1,17                   0,00
ALIANZA LIBRE 0,02            0,84                 1,85                   0,00
MAS        0,04            1,87                 4,08                   93,10
UNIDAD     0,03            1,10                 2,90                   6,90
Probabilidad de segunda vuelta: 69,80%
--------------------------------------------
```

Y asi para todos, podra ver esto al ejecutar el .jar que estara apenas habra el disco como TssTarea4.jar

Conclusión

El programa es capaz de reproducir los resultados requeridos tanto en Excel como en Java, lo cual es interesante dándonos pronósticos con datos relevantes.