Voting System Documentation

This document provides a comprehensive overview of the Voting System project, which includes the following classes:

- `VotingSystem`
- `Candidate`
- `VoteRecord`
- `Voter`
- `ResultPanel`
- `VotingPanel`

Class Details

`VotingSystem`

The

`VotingSystem`

class manages the overall voting process. It handles the registration of candidates and voters, the casting of votes, and the retrieval of election results.

Key Features and Functionality

- Manages candidates and voters.
- Starts and ends the election.
- Allows voters to cast votes.
- Retrieves vote results.

Code Snippet:

```
package models;

import java.util.*;

public class VotingSystem {

    private Map<String, Candidate> candidates;
    private Map<String, Voter> voters;
    private List<VoteRecord> votes;
    private boolean electionStarted;

    public VotingSystem() {
        this.candidates = new HashMap<>();
        this.voters = new HashMap<>();
        this.votes = new ArrayList<>();
        this.electionStarted = false;
    }
    public void addCandidate(Candidate candidate) {
        this.candidates.put(candidate.getId(), candidate);
    }

    public void addVoter(Voter voter) {
        this.voters.put(voter.getId(), voter);
    }

    public void startElection() {
        this.electionStarted = true;
    }

    public void endElection() {
        this.electionStarted = false;
    }

    public void castVote(String voterId, String candidateId) {
        if (electionStarted && voters.containsKey(voterId) && candidates.containsKey(cand
            VoteRecord vote = new VoteRecord(voterId, candidateId);
            votes.add(vote);
        }
    }

    public Map<String, Integer> getVoteResults() {
      Map<String, Integer> results = new HashMap<>();
        for (VoteRecord vote : votes) {
            String candidateId = vote.getCandidateId();
            results.put(candidateId, results.getOrDefault(candidateId, 0) + 1);
        }
        return results;
    }

    public Map<String, Candidate> getCandidates() {
        return candidates;
    }

    public Map<String, Voter> getVoters() {
        return voters;
    }

    public List<VoteRecord> getVotes() {
        return votes;
    }

    public boolean isElectionStarted() {
        return electionStarted;
    }
}
```

`Candidate`

The

```
Candidate
```

class represents a candidate in the election. It stores the candidate's ID, name, and party affiliation.

Code Snippet:

```
package models;
```

```
public class Candidate {
    private String id;
    private String name;
    private String party;

    public Candidate(String id, String name, String party) {
        this.id = id;
        this.name = name;
        this.party = party;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String getParty() {
        return party;
    }
}
```

`VoteRecord`

The

`VoteRecord`

class represents a single vote cast in the election. It stores the ID of the voter and the ID of the candidate they voted for.

Code Snippet:

```
package models;

public class VoteRecord {
    private String voterId;
    private String candidateId;

    public VoteRecord(String voterId, String candidateId) {
        this.voterId = voterId;
        this.candidateId = candidateId;
    }

    public String getVoterId() {
        return voterId;
    }

    public String getCandidateId() {
        return candidateId;
    }
}
```

`Voter`

The

`Voter`

class represents a person who is eligible to vote in the election. It stores the voter's ID and name.

Code Snippet:

```
package models;

public class Voter {
    private String id;
    private String name;

    public Voter(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```

`ResultPanel`

The

`ResultPanel`

class is a Swing component used to display election results in a graphical user interface.

Code Snippet:

```
package views;

import models.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.Map;
import java.util.HashMap;
import java.util.List;

public class ResultPanel extends JPanel {

    private static final long serialVersionUID = 1L;
    private VotingSystem system;
    private JTextArea resultArea;

    public ResultPanel(VotingSystem system) {
        this.system = system;
        setLayout(new BorderLayout());
        resultArea = new JTextArea();
        resultArea.setEditable(false);
        resultArea.setFont(new Font("Monospaced", Font.PLAIN, 14));

        JScrollPane scrollPane = new JScrollPane(resultArea);
        add(scrollPane, BorderLayout.CENTER);

        JButton refreshBtn = new JButton("Refresh Results");
        refreshBtn.addActionListener(e -> refreshResults());
        add(refreshBtn, BorderLayout.SOUTH);

        refreshResults();
    }

    private void refreshResults() {
        try {
            Map<String, Integer> results = getVoteResults();
            Map<String, Candidate> candidates = system.getCandidates();

            StringBuilder sb = new StringBuilder();
            sb.append("ELECTION RESULTS\\n");
            sb.append("================\\n\\n");

            if (results.isEmpty()) {
                sb.append("No votes have been cast yet.\\n");
            } else {
                sb.append(String.format("%-20s %-15s %-10s\\n", "Candidate", "Party", "Vo
                sb.append(String.format("%-20s %-15s %-10s\\n", "---------", "-----", "--

                for (Map.Entry<String, Integer> entry : results.entrySet()) {
                    Candidate c = candidates.get(entry.getKey());
                    String partyName = (c != null) ? c.getParty() : "N/A";
                    sb.append(String.format("%-20s %-15s %-10d\\n",
                            c.getName(), partyName, entry.getValue()));
                }

                int totalVotes = results.values().stream().mapToInt(Integer::intValue).su
                sb.append("\\nTotal votes cast: ").append(totalVotes).append("\\n");
            }

            resultArea.setText(sb.toString());
        } catch (Exception ex) {
            resultArea.setText("Error loading results: " + ex.getMessage());
        }
    }

    private Map<String, Integer> getVoteResults() {
        List<VoteRecord> votes = system.getVotes();
        Map<String, Integer> results = new HashMap<>();

        if (votes != null) {
            for (VoteRecord vote : votes) {
                String candidateId = vote.getCandidateId();
                results.put(candidateId, results.getOrDefault(candidateId, 0) + 1);
        }
    }
        return results;
    }
}
```

`VotingPanel`

The

```
VotingPanel
```

class is a Swing component that provides the user interface for casting votes.

Code Snippet:

```
package views;

import models.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.List;
import java.util.Map;

public class VotingPanel extends JPanel {

    private static final long serialVersionUID = 1L;
    private VotingSystem system;
    private String voterId;

    public VotingPanel(VotingSystem system, String voterId) {
        this.system = system;
        this.voterId = voterId;
        setLayout(new BorderLayout());

        if (!system.isElectionStarted()) {
            add(new JLabel("The election has not started yet. Please check back later."),
            return;
        }

        if (!system.getVoters().containsKey(voterId)) {
            add(new JLabel("Invalid Voter ID.  You are not registered to vote.", SwingCon
            return;
        }

        if (hasVoted()) {
            add(new JLabel("You have already voted. Thank you for participating!", SwingC
            return;
        }

        ButtonGroup group = new ButtonGroup();
        JPanel optionsPanel = new JPanel(new GridLayout(0, 1));

        Map<String, Candidate> candidates = system.getCandidates();

        if (candidates.isEmpty()) {
            add(new JLabel("No candidates are available to vote for.", SwingConstants.CEN
            return;
        }
        for (Candidate candidate : candidates.values()) {
            JRadioButton radio = new JRadioButton(candidate.getName() + " (" + candidate.
            radio.setActionCommand(candidate.getId());
            group.add(radio);
            optionsPanel.add(radio);
        }

        add(new JScrollPane(optionsPanel), BorderLayout.CENTER);

        JButton voteButton = new JButton("Cast Vote");
        voteButton.addActionListener(this::castVote);
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(voteButton);
        add(buttonPanel, BorderLayout.SOUTH);
    }

    private void castVote(ActionEvent e) {
        ButtonGroup group = getButtonSelection();
        if (group == null || group.getSelection() == null) {
            JOptionPane.showMessageDialog(this, "Please select a candidate to vote for.")
            return;
        }

        String candidateId = group.getSelection().getActionCommand();
        system.castVote(voterId, candidateId);
        JOptionPane.showMessageDialog(this, "Vote cast successfully! Thank you for voting

        removeAll();
        add(new JLabel("You have successfully voted. Thank you for participating!", Swing
        revalidate();
        repaint();
    }

    private ButtonGroup getButtonSelection() {
        Component[] components = getComponents();
        for (Component component : components) {
            if (component instanceof JScrollPane) {
                JScrollPane scrollPane = (JScrollPane) component;
                Component[] innerComponents = scrollPane.getViewport().getComponents();
                for (Component innerComponent : innerComponents) {
                    if (innerComponent instanceof JPanel) {
                        JPanel panel = (JPanel) innerComponent;
                        ButtonGroup group = getButtonGroup(panel);
                        if (group != null) {
                            return group;
                        }
                    }
                }
            }
        }
        return null;
    }

    private ButtonGroup getButtonGroup(JPanel panel) {
        ButtonGroup foundGroup = null;
        Component[] components = panel.getComponents();
        for (Component component : components) {
            if (component instanceof JRadioButton) {
                JRadioButton radio = (JRadioButton) component;
                ButtonModel model = radio.getModel();
                if (model.getGroup() instanceof ButtonGroup) {
                    ButtonGroup group = model.getGroup();
                    if (foundGroup == null) {
                        foundGroup = group;
                    } else if (foundGroup != group) {
                        return null;
                    }
                } else {
                    return null;
                }
            }
        }
        return foundGroup;
    }

    private boolean hasVoted() {
        List<VoteRecord> votes = system.getVotes();
        if(votes == null){
            return false;
        }
        for (VoteRecord vote : votes) {
            if (vote.getVoterId().equals(voterId)) {
                return true;
            }
        }
        return false;
    }
}
```