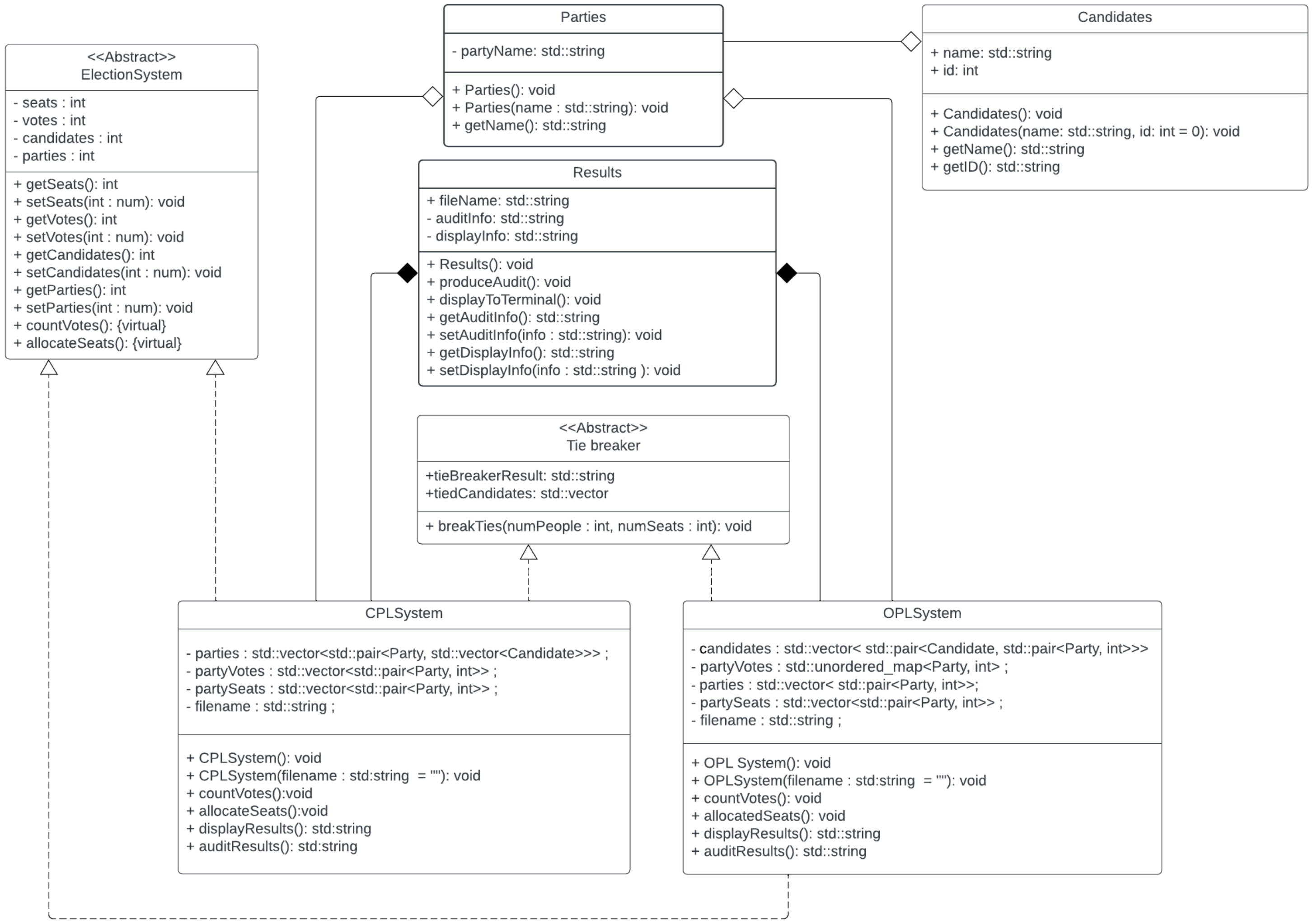
Team 7

SDD UML Class Diagram


Parties
- partyName: std::string
+ Parties(): void
+ Parties(name : std::string): void
+ getName(): std::string
Candidates
+ name: std::string
+ id: int
+ Candidates(): void
+ Candidates(name: std::string, id: int = 0): void
+ getName(): std::string
+ getID(): std::string
<<Abstract>>
ElectionSystem
- seats : int
- votes : int
- candidates : int
- parties : int
+ getSeats(): int
+ setSeats(int : num): void
+ getVotes(): int
+ setVotes(int : num): void
+ getCandidates(): int
+ setCandidates(int : num): void
+ getParties(): int
+ setParties(int : num): void
+ countVotes(): {virtual}
+ allocateSeats(): {virtual}
Results
+ fileName: std::string
- auditInfo: std::string
- displayInfo: std::string
+ Results(): void
+ produceAudit(): void
+ displayToTerminal(): void
+ getAuditInfo(): std::string
+ setAuditInfo(info : std::string): void
+ getDisplayInfo(): std::string
+ setDisplayInfo(info : std::string): void
<<Abstract>>
Tie breaker
+tieBreakerResult: std::string
+tiedCandidates: std::vector
+ breakTies(numPeople : int, numSeats : int): void
CPLSystem
- parties : std::vector<std::pair<Party, std::vector<Candidate>>> ;
- partyVotes : std::vector<std::pair<Party, int>> ;
- partySeats : std::vector<std::pair<Party, int>> ;
- filename : std::string ;
+ CPLSystem(): void
+ CPLSystem(filename : std:string = ""): void
+ countVotes():void
+ allocateSeats():void
+ displayResults(): std:string
+ auditResults(): std:string
OPLSystem
- candidates : std::vector< std::pair<Candidate, std::pair<Party, int>>>
- partyVotes : std::unordered_map<Party, int> ;
- parties : std::vector< std::pair<Party, int>>;
- partySeats : std::vector<std::pair<Party, int>> ;
- filename : std::string ;
+ OPL System(): void
+ OPLSystem(filename : std:string = ""): void
+ countVotes(): void
+ allocatedSeats(): void
+ displayResults(): std::string
+ auditResults(): std::string

Notes:

- “Parties” & “Candidates” can exist regardless of whether CPL or OPL elections exist or not.
- “Results” cannot exist without the CPL or OPL election system taking place.
- Candidate has an ID in case there are two candidates with the same name. This is only used in OPL elections, where the candidates get the votes, and they will be sorted in order of votes ascending.
- The CPL and OPL classes realize from the ElectionSystem abstract class

Overview:

Each voting system stores the votes as needed for their election-type constraints. We will explain the data structures of both the CPL and OPL election types to clear up any confusion.

In CPL, we read the parties in order from the ballot file. When votes are counted, we count the index position of the vote per each ballot that was scanned. This way, if the first ballot had a vote at index 0, then the first party gets that vote, and so on. We use a vector of pairs, “partyVotes”, which keeps track of all the parties, and each party gets their votes stored inside the second element of that pair. This “partyVotes” vector can then be sorted based on the second element in the pair. In the CPL system, we have a vector of pairs, “parties”, where each pair has a party and a vector of all its candidates. We believe this will help us easily allocate seats to the winners by indexing through each Party of “parties”, and selecting as many Candidates as the Party has seats, starting from Candidate 0 until we run out of seats for that party. We also need to store the number of seats each party gets to do this. We have one last vector of pairs “partySeats”, which allows us to give each party a certain number of seats based on the number of votes they got.

A similar process happens in the OPL election but from a different starting point. The first data structure we have, “candidates”, stores candidate information. We use a vector of pairs, where each Candidate (first element) has their own pair (second element), which contains party affiliation (party name), and this candidate's number of votes. After counting all of the votes, we can sort “candidates” based on the number of votes. After this, we would go through this vector, count the number of votes each party has, and store this info in “partyVotes”. We want to be able to easily sort the parties by votes, so we move this info into a vector called “parties”. Our last vector, “partySeats”, which stores the amount of seats each party has earned, is filled out after we allocate seats. To be able to assign winning candidates, we can say for example that the party Green has 3 seats, so we go through our “candidates” list, and find the first 3 candidates that have a Party which has the name of “Green”.