`abstract VoteCounter:`

`name` is a String field initialized from the constructor with getter `getName()`.

`voteCount` is an int field initialized 0, getter `getVoteCount()` and setter `setVoteCount(int)`.

`addVoteCount(n)` adds `n` to `voteCount`.

`Candidate extends VoteCounter:`

`hasSeat` is a boolean field initialized `false` with mutator `giveSeat()` and getter `isSat()`

`Party extends VoteCounter:`

`indep` is a boolean field initialized `false` with mutator `setIndep(bool)` and getter `isIndep()`

```
abstract Election:
int[] tiebreak(n, r): // n choose r
```

Initialize `int[r] selected` array to value of `-1` everywhere

While `i < selected.length:`

Select a random integer `k` from `0` (inclusive) to `n` (exclusive)

If `k` is not in `selected`, store `k` in `selected[i]` and increment `i`

Return `selected`

```
abstract processBallotData(lines)
abstract allocateSeats()
abstract generateAuditFile()
abstract displayWinners()
```

```
OPLElection:
```

`processBallotData(lines):`

set `seatCount` equal to `lines[1]`

set `ballotCount` equal to `lines[2]`

set `partyCount` equal to `lines[3]`

initialize `party[]` to have size `partyCount`

initialize `candidates` as an array or array list of candidates with size `partyCount`

for each candidate:

separate the party from the associated line

append the remaining token from that line to candidates for the row of that party

for each ballot:

increment the votecount for the proper candidate

for each candidate:

increment votecount for the associated party by the candidate's vote count

`allocateSeats():`

initialize `allocationData` to 5 by `partycount`

set first column of array to the `voteCount` of the respective party

set `votesPerSeat` to ballotCount/seats

divide `voteCount` of EachParty by `votesPerSeat` and put into column representing First allocation of seats (column 2)

subtract first allocation of seats times `votesPerSeat` from `voteCount` to get remaining votes (column 3)

subtract seats from each value in first allocation of seats and set the result into remainingSeats

divide remainingSeats by partyNumber and set each party's second allocation of seats equal to it (column 4)

set remainingSeats equal to remainingSeats modulo partyNumber

increment second allocation of seats for each party for the remaining seats number of party largest parties by remaining votes.

if there is a tie, call tiebreak

add second allocation of seats with first allocation of seats to get Final Seat total (column 5)

`generateAuditFile():`

Gather `partyCount, ballotCount, seatCount` along with `allocationData` and the `parties` and `candidates` lists from class fields.

Calculate percentage of votes and proportion of seats.

Format data (including election type) using `StringBuilder` and write to file with a filename involving the system time.

`displayWinners():`

print type of election, the number of seats, the party number, and the number of ballots

`loop` through all candidates, printing all that won a seat

`loop` through candidates again, this time printing all candidates, the number of votes received, the percentage of votes received, and whether they won a seat or not

CPLElection:

processBallotData(lines):

set seatCount equal to lines[1]

set ballotCount equal to lines[2]

set partyCount equal to lines[3]

initialize party[] to have size partyCount

initialize candidates as an array or array list of candidates

for each party:

separate the party from the associated line

append the remaining tokens from that line to candidates

for each ballot:

increment the voteCount for the proper candidate

for each candidate:

increment voteCount for the associated party by the candidate's voteCount

allocateSeats():

initialize allocationData to 5 by partycount

set first column of array to the voteCount of the respective party

set votesPerSeat to ballotCount/seats

divide voteCount of EachParty by votesPerSeat and put into column representing First allocation of seats (column 2)

subtract first allocation of seats times votesPerSeat from voteCount to get remaining votes (column 3)

subtract seats from each value in first allocation of seats and set the result into remainingSeats

divide remainingSeats by partyNumber and set each party's second allocation of seats equal to it (column 4)

set remainingSeats equal to remainingSeats modulo partyNumber

increment second allocation of seats for each party for the remaining seats number of party largest parties by remaining votes.

if there is a tie, call tiebreak

add second allocation of seats with first allocation of seats to get Final Seat total (column 5)

generateAuditFile():

Gather `partyCount, ballotCount, seatCount` along with `allocationData` and the `parties`.

Calculate percentage of votes and proportion of seats.

Format data (including election type) using `StringBuilder` and write to file with a filename involving the system time.

`displayWinners():`

print type of election, the number of seats, the party number, and the number of ballots

`loop` through all candidates, printing all that won a seat

`loop` through party, printing the number of votes received, the percentage of votes received, and how many seats they won

---

`AESDriver:`

`main ():`

Initialize `FileInput` object

Set `lines` equal to `FileInput.getLines()`

if `is(OPL):`

Initialize `Election` object to OPLElection

else:

Initialize `Election` object to CPLElection

Call `Election.processBallotData(lines)`

Call `Election.allocateSeats()`

Call `Election.generateAuditFile()`

Call `Election.displayWinners()`

`FileInput:`

`String filename`

`String[] lines field has getter getLines`

`FileInput ():`

Call `promptFilename()`

While not `validateFile()`, give error message and call `promptFilename()`

`FileInput (String):`

Set `filename` equal to `String`

If not `validateFile()`, exit program with error message

```
promptFilename ():
      Ask for Filename
      read user input
`     Set filename equal to user input
validateFile ():
     try (open File):
          if (Not Can read file):
               print error about file permissions
               return false
          return true
     catch (file open failed):
          print error about file not existing
          return false

isOPL():
     return lines[0].equals("OPL")

getLines():
     return lines;
```