# Assignment 4

*due 22/3/2014*

1. This assignment is based on an old exam question and involves developing a vote-counting application for a simple electoral system.

2. Suppose that we wish to complete a vote-counting application for a simple electoral system based in single-seat constituencies. There are a number of candidates, of whom one is to be elected. Each voter ranks the candidates in order of preference $1, 2, 3$ *etc.* on a ballot paper. During the count, each ballot paper is distributed initially to the candidate who received the highest preference on that paper. Once this is done, candidates are eliminated one by one until only one remains, the last remaining candidate being deemed the winner. At each elimination the candidate with the lowest number of votes is chosen and his votes are re-distributed among the remaining candidates, with each voting paper being awarded to the remaining candidate with the highest preference on that ballot paper. All ties are broken by the drawing of lots.

3. The application involves four main classes: Candidate, BallotPaper and VoteCounter that you will have to implement from scratch; their behaviours are described below. The completed application may also rely on various ADTs (Lists, Maps and so on).

   Class Candidate simply records the details of an individual candidate (name, party). It must support the following methods: setName, getName, setParty and getParty. [1]

   Class BallotPaper captures the idea of the ballot paper completed by one voter. It records the order in which the voter ranks the candidates. We assume that there are no spoilt ballots and that each voter dutifully ranks all $n$ candidates in order of preference from one (highest preference) to $n$ (lowest preference). BallotPaper objects support the following operations.

   - **setPreference(n, c):** Set the n-th preference for candidate c. *Input:* int, Candidate; *Output:* None.
   - **getPreference(n):** Return the candidate who received the n-th preference on this voting paper. *Input:* int; *Output:* Candidate.

   Class VoteCounter encapsulates the mechanics of the vote-counting process. Your class must incorporate a mechanism that records the votes (BallotPaper objects) cast for each candidate. You may assume that the constructor of VoteCounter takes as its argument a list of the candidates (provided in the form List<Candidate>) and with each candidate initially having no votes.

   Your implementation of VoteCounter will need to support the following method (and probably several others in addition).

```
/**
*  Determine the outcome of the election based on the votes in
*  votesCast. Return the  winning candidate.
*/
public Candidate determineResult(List<BallotPaper> votesCast)
```

[1] For this assignment we will assume that candidate names are unique.

Note that the bundle of votes cast in the election are provided as a List<BallotPaper> object. The above method should generate a detailed summary of the progress of the count at each stage as well as determining the eventual winner.

4. Notes

   - Provide implementations for classes Candidate, BallotPaper and VoteCounter. You may make use of any of the ADT implementations that are available on the webpage *e.g.* ArrayBasedMap, if required. You may also find that you need to implement some additional minor classes.
   - You must test your application as rigorously as you can. This is not an optional extra, this is a key requirement. The testing code should be in one or more independent classes named Test1.java, Test2.java and so on.
   - Submit your completed application as a `a4.tgz` file using Moodle. The directory must include all the files your application requires to compile and to execute (including ADT implementations, test classes and so on).