Part A

* **Nodes** are stored within a HashMap, where key is the ID of the node. This reduces the time it takes to look up Nodes. Each node maintains a reference to its next and previous neighbour, as well as a reference to a list of all its neighbours. In Part A, only “next” and “previous” are used, since messages travel in a ring around the network.
* **Elections** for a specific round are stored as Action instances within a HashMap, where key is the round number. In this way, all actions corresponding to the current round can be executed. Note that the custom Action type is purely for extensibility purposes. It is only used for elections now.
* In each round, the latest added message is collected from each node’s outgoing messages queue. Each of these messages is then delivered to the „next” neighbour of the specific node.
* Each node maintains a “finished” flag that indicates whether the node has any remaining messages in its incoming or outgoing messages queue. This is to avoid terminating the algorithm early in case the node-threads could not keep up with the pace of the network’s (main) thread. For example, if a node takes too long converting an incoming message to an outgoing message, the “finished” attribute would flag “false”, and the algorithm would not terminate just because there are no messages to send in that specific round.
* Once an election is complete, the leader sends a leader message to its next neighbour. This message then has to visit all nodes until it comes back to the leader. Each node sets itself as non-participant.
* All election and leader messages include the ID of the node that initially released the message. This is for debugging purposes, since there can be multiple elections happening at the same time.