Part A

Setup

* **Nodes** are stored within a *HashMap<Integer, Node>,* 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 *nextNode* and *previousNode* are used, since messages travel in a ring around the network.
* **Elections** for a specific round are stored as Action instances within a *HashMap<Integer, ArrayList<Action*>>, 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.
* **Failures** are stored in a separate list since they are not related to a specific round.

Rounds

* In each round, the least recently added message is collected from each node’s outgoing messages queue. Each of these messages is then delivered into the incoming messages queue of *nextNode* of the specific node. Delay for each round is simulated by thread sleep.
* 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 when receiving a leader message.
* All election and leader messages include the ID of the node that initially released the message. This is for debugging purposes, as there can be multiple elections happening at the same time.
* The **termination** condition is when the message buffer of the network is empty, the action buffer is empty, all nodes have finished processing, and the list of failures is empty.

Part B

Failures

* After all regular messages have been delivered, and all actions have been executed, we start failures at most one per round.
* When a node fails, all neighbours of the failed node are notified by the network, so that they can rearrange their *nextNode*, *previousNode*, and *neighbours* references appropriately: the failed node is removed from the neighbours list of each neighbour; the *nextNode* reference of the failed node’s *previousNode* is set to the failed node’s *nextNode*; and similarly, the *previousNode* reference of the failed node’s *nextNode* is set to the failed node’s *previousNode*.
* If the failed node was a leader, all neighbours, which got notified of the failure, start a new election.
* If all but one nodes fail, the *nextNode* and *previousNode* references of the last node get set to null, and the algorithm stops.