Testing Document Assignment 3

Nithin D (a1943553)

Paxos Protocol

Understanding implementation before moving ahead with test case:

Key Methods in Context

1. **createNode**:
   - Instantiates a Paxos node with specified parameters:
     - id: Unique identifier for the node.
     - type: Role of the node (proposer or acceptor).
     - Flags for delays or offline states.
     - Initial nominee ID (if applicable).
   - This method encapsulates node configuration and ensures each node is initialized correctly for testing scenarios.
2. **startNode**:
   - Starts the Paxos node, enabling it to participate in the consensus process.
   - Ensures the node is active and ready to handle proposals and responses as part of the Paxos protocol.
3. **setPreferredNominees**:
   - Configures a node's voting behavior by specifying its preferred nominees.
   - This is critical for defining how nodes interact during elections, especially in scenarios with simultaneous proposals or crashes.
4. **waitForCompletion**:
   - Pauses test execution for a specified duration to allow the Paxos process to complete.
   - Necessary to accommodate asynchronous communication and delays in complex setups.

Configurable Parameters in createNode method:

In the Paxos implementation, the behavior of each node can be customized to simulate real-world scenarios such as delays in communication or dropped messages. This is achieved through the parameters passed to the createNode method when setting up a node. Here’s how the parameters are used in the test cases:

1. **Short Delay (shortDelay)** :
   - If set to true, the node simulates a small (two second) delay in responding to Paxos messages.
   - This mimics network latency or slight processing delays.
   - Example:
     - In **Test Case 3**, node m2 is configured with shortDelay to introduce minor latencies during voting queries.
2. **Long Delay (longDelay)**:
   - If set to true, the node simulates a substantial (ten second) delay in responding.
   - This represents scenarios like heavily congested networks or slow processing due to high load.

   - Example:
     - In **Test Case 3**, node m3 is configured with longDelay, forcing other nodes to wait longer for responses, testing the robustness of the Paxos protocol.
3. **Dropped Messages (dropMessage)**:
   - If set to true, the node drops some or all Paxos messages.
   - This is used to simulate network delay or unresponsive nodes.
   - Example:
     - In **Test Case 3**, node m2 is configured with dropMessage to represent a node going offline or becoming unreachable during the election process.

How These Parameters Enhance Testing:

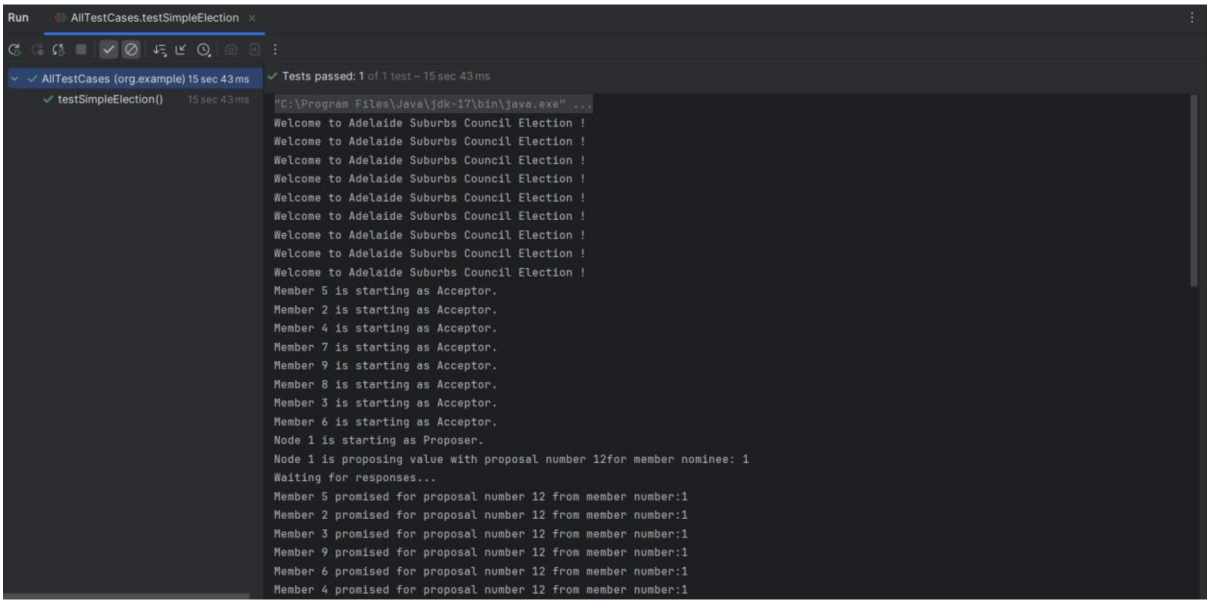
- **Realism**: These parameters replicate practical challenges such as latency, message loss, and node crash, ensuring the implementation is robust under diverse conditions.
- **Fault Tolerance**: They stress-test the Paxos protocol's ability to maintain correctness and liveness despite disruptions.
- **Customizability**: By combining these parameters, we can simulate complex scenarios, such as a network with high latency and partial node crash.

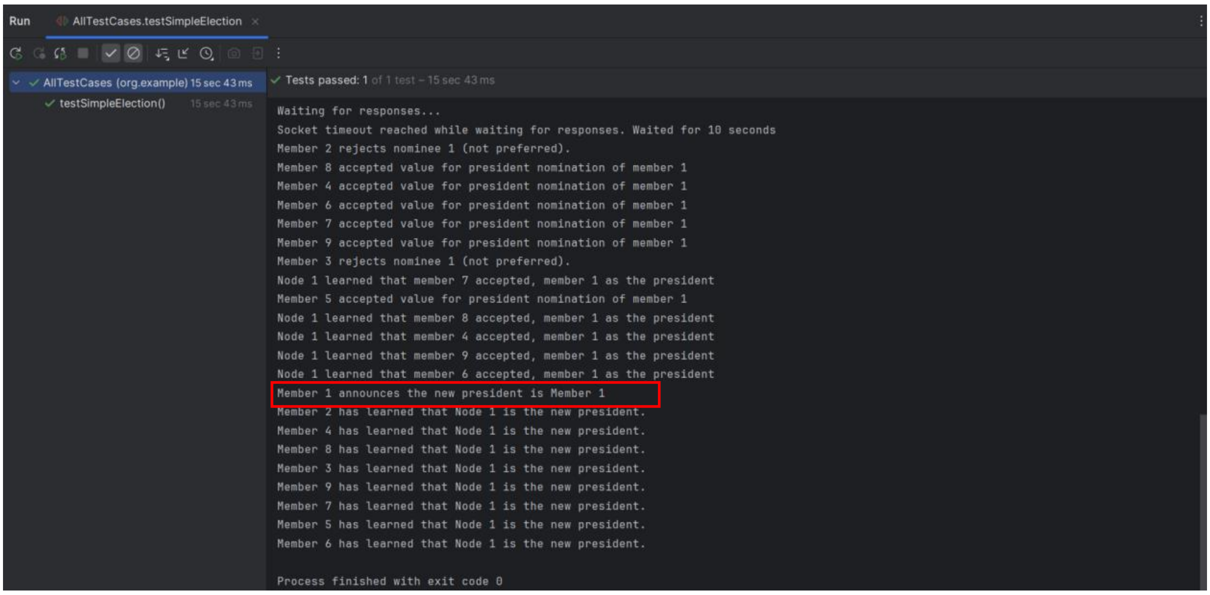
Test Cases

Test Case 1: testSimpleElection(): Simple Election: 30 points – Paxos implementation works in the case where all M1-M9 have immediate responses to voting queries.

Test Case 1: Simple Election

- **Objective**: Verify the correctness of the Paxos implementation in a straightforward scenario where all nodes respond immediately to voting queries.
- **Setup**:
  - m1 acts as the proposer and nominates itself for election. Other nodes (m2 to m9) are configured as acceptors.
  - The proposer (m1) explicitly sets itself as the preferred nominee (setPreferredNominees), ensuring it will only vote for itself.
  - Nodes m2 and m3 are also configured with their respective preferences, though in this test, m1 is designed to succeed.
- **Execution**:
  - Nodes participate in the Paxos election process, with responses being immediate.
  - The test waits for up to 15 seconds to allow the election to complete.
- **Validation**:
  - Checks that m1 is elected as the president (getCurrentPresident), confirming that the Paxos protocol operates correctly in this simple, synchronous scenario.



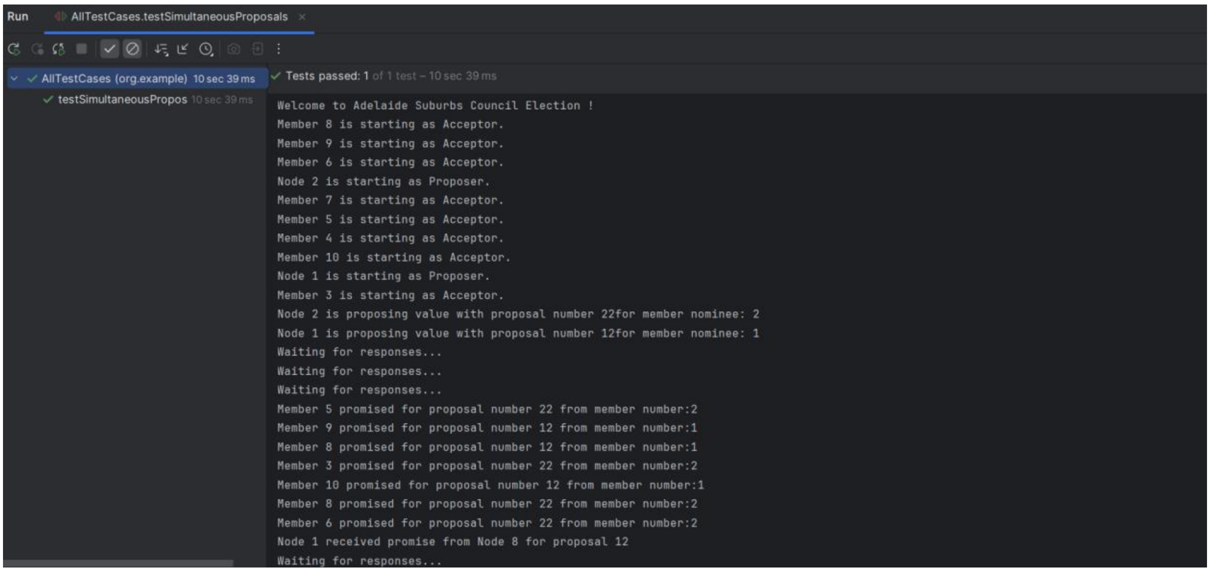


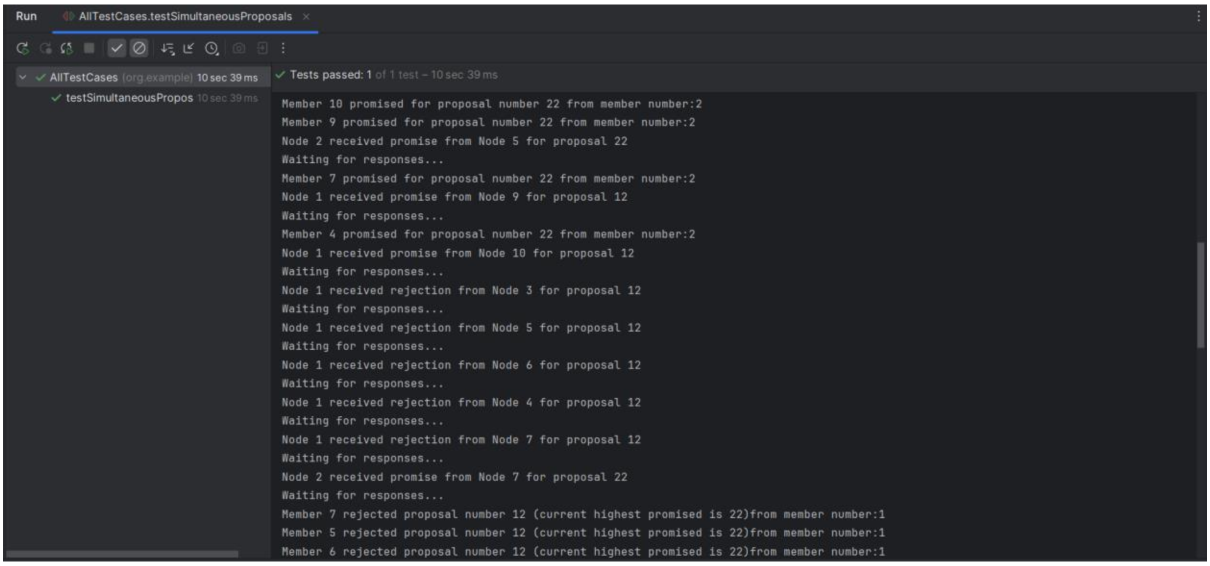
Here, you can observe, Member 1 after consensus was reached announces to all the members the new president, including to the members who rejected the nominee.

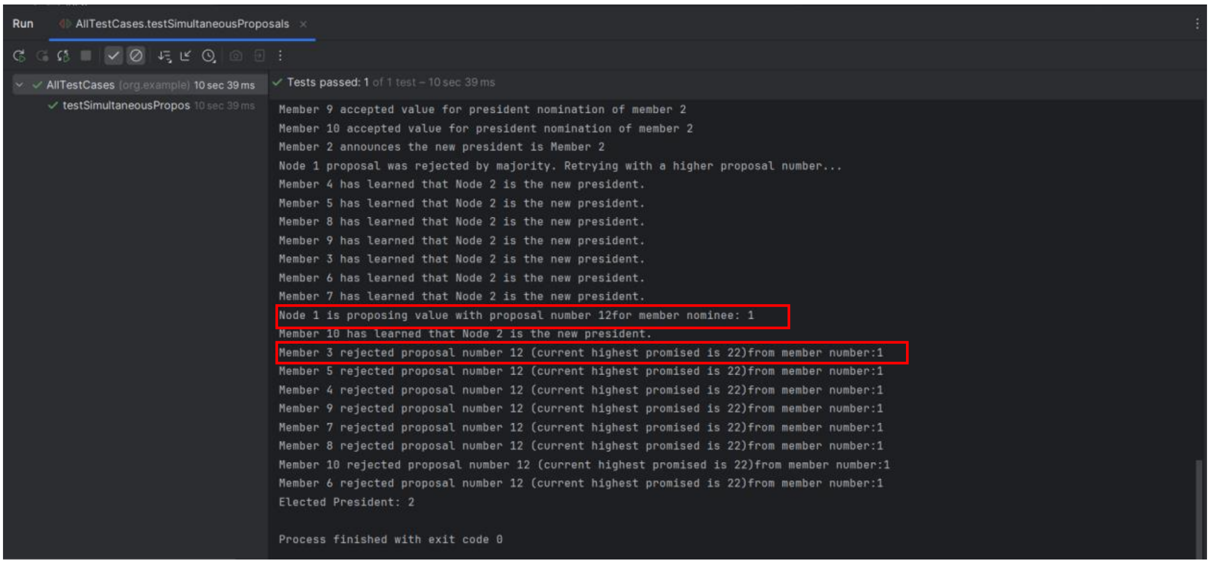
Test Case 2: testSimultaneousProposals(): 10 points - Paxos implementation works when two councillors send voting proposals at the same time.

Test Case 2: Simultaneous Proposals

- **Objective**: Validate Paxos behavior when two proposers (m1 and m2) simultaneously propose themselves for election.
- **Setup**:
  - Both m1 and m2 are proposers, and each nominates itself. Other nodes (m3 to m9) are acceptors.
  - Proposers set their preferred nominees to themselves using setPreferredNominees, while acceptors (e.g., m3) are configured similarly but are passive participants.
- **Execution**:
  - Two proposals propagate simultaneously, leading to contention for leadership.
  - The system waits for 10 seconds to allow the election to complete.
- **Validation**:
  - Checks that the Paxos protocol resolves the simultaneous proposals to elect one of the proposers (m1 or m2) as president.
  - Ensures that only one leader is elected, demonstrating the system's ability to handle conflicts.







Here, you can notice the importance of proposal id, each proposal has unique id. So, when Node/Member 1 made a proposal with number 12, the other Node/Members in the protocol rejected it, as the proposal number was lower than the current proposal number which was 22 from Member 2. Hence, member 2 was the president.

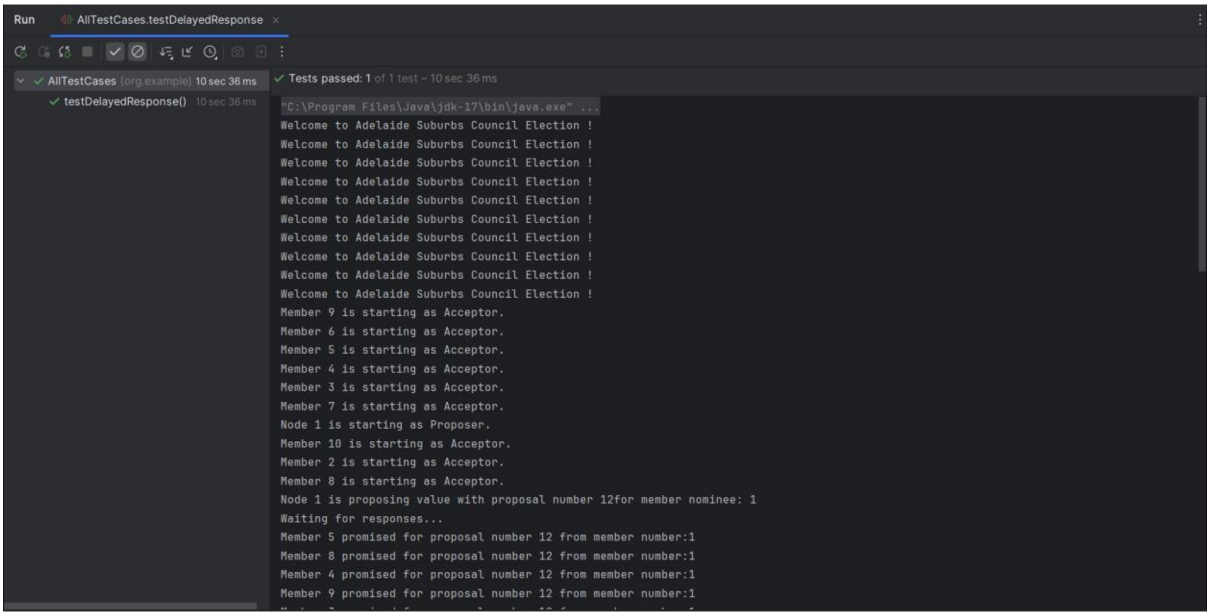
Test Case 3: testDelayedResponse(): 30 points – Paxos implementation works when M1 – M9 have responses to voting queries suggested by several profiles (immediate response, small delay, large delay and no response), including when M2 or M3 propose and then go offline.

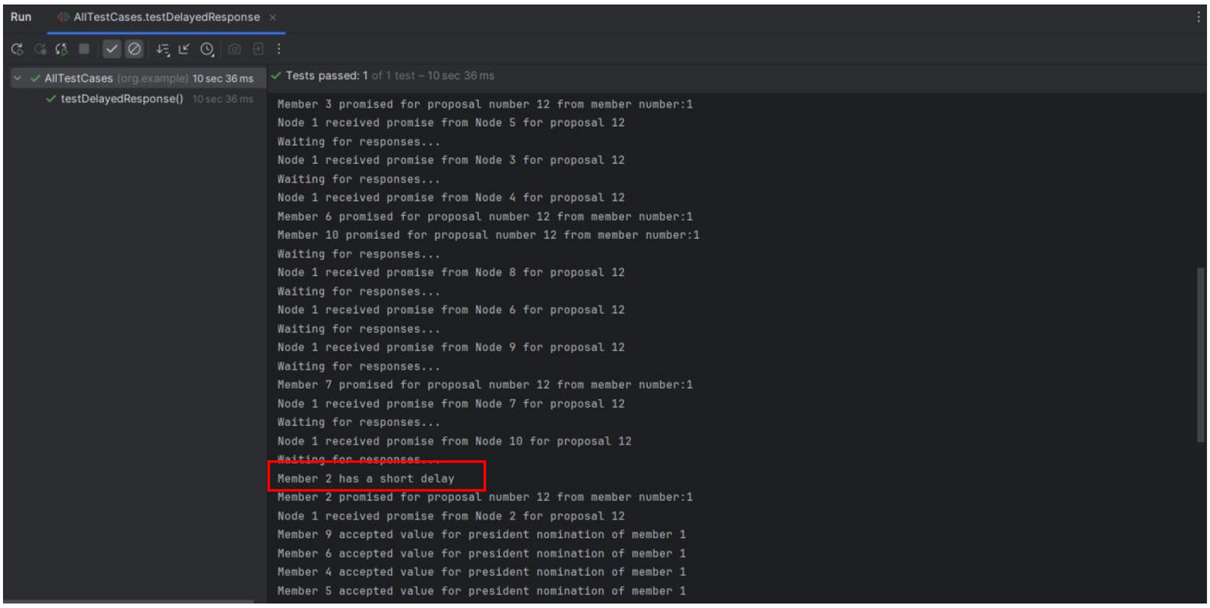
Test Case 3: Delayed Response

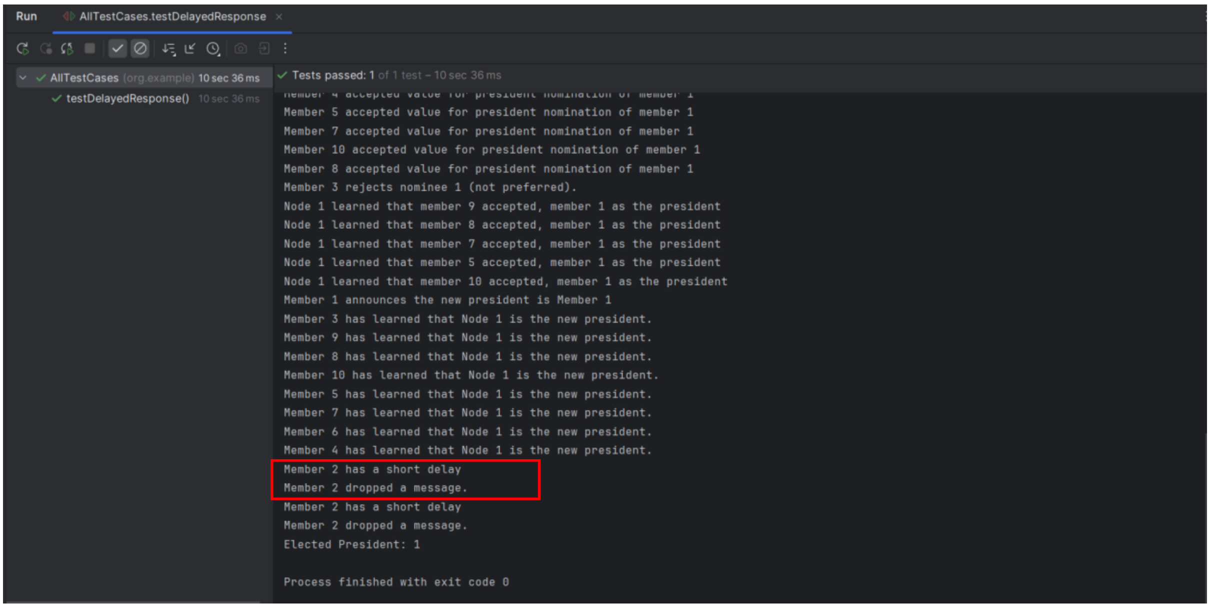
- **Objective**: Test Paxos implementation under various response conditions, including delays and node crashes.
- **Setup**:

- m1 is the proposer and nominates itself. Nodes m2 and m3 are acceptors configured with:
  - m2: Simulates a delayed response and eventual offline state.
  - m3: Simulates intermittent delays during the election.
- Remaining nodes (m4 to m9) are configured as standard acceptors without delays or crashes.
- Each node votes according to its setPreferredNominees configuration, prioritizing itself.

- **Execution**:
  - Proposer initiates the Paxos protocol, and the system handles responses with varying latencies and crashes.
  - The test allows a 10-second window for the election to complete.
- **Validation**:
  - Verifies that the protocol successfully elects m1 as the president, despite the presence of delays and node crashes, demonstrating the fault tolerance and resilience of the Paxos implementation.







Here, you can notice the delay in the system, I have set 2 second for short delay and 10 second for long delay. The response from member 2 is delayed but still the protocol executes as intended.

Test Case 4: testOnly5MembersAreAvailable(): No concensus can be reached, more than 5 members are unavailable.

- **Scenario**: Only 5 nodes (m1 to m5) are operational, less than the quorum needed for consensus.
- **Setup**:
  - m1 is the proposer; other nodes are acceptors.
  - Some nodes have voting preferences; others accept any proposals.
- **Execution**:
  - m1 sends a proposal and waits for responses.

- Insufficient promises are received due to lack of quorum.
- **Outcome**:
    - Consensus cannot be reached, and no president is elected.
    - Key Print:
        - "Socket timeout reached while waiting for responses."
        - "Not enough members are available, consensus could not be reached."
- **Validation**: m1.getCurrentPresident() returns null, as expected per Paxos rules.