Assignment Title: Simulating High Availability with Load Balancing in Java

---

Assignment Description:

Availability ensures that services and resources are accessible to authorized users whenever needed, even in the face of high demand or partial system failures. In this assignment, you will implement a Java program that simulates a simple load balancer to distribute requests across multiple servers, ensuring high availability.

Your program will:

1. Simulate a set of servers that handle requests.
2. Implement a load balancer to distribute incoming requests across these servers.
3. Monitor server health and reallocate requests dynamically if a server becomes unavailable.

Requirements:

1. **Input and Output**:
    - Prompt the user to specify the number of servers and the total number of incoming requests.
    - Simulate request processing and output which server handles each request.
    - Display server status (active or inactive) during the simulation.
2. **Load Balancer**:
    - Implement a round-robin algorithm to distribute requests across servers.
    - Handle server failure by detecting unavailable servers and redistributing their requests to active servers.
3. **Server Simulation**:
    - Each server should have a unique ID and a health status (active or inactive).
    - Simulate random server failures during the program's runtime.
4. **Resilience and Recovery**:
    - Implement logic to dynamically reallocate requests if a server fails.

- Include an option to simulate server recovery and reintegrate recovered servers into the load balancer.

5. **Documentation and Testing**:
    - Include comments explaining the functionality of each part of the code.
    - Provide at least three test cases to demonstrate the program's ability to maintain availability under various conditions.

Deliverables:

1. **Java Source Code**:
    - Submit a .java file or multiple files if necessary, ensuring code is well-structured and documented.
2. **Test Results**:
    - Submit a document summarizing the test cases, including the number of servers, incoming requests, server failures, and the observed output.
3. **Optional Enhancements** (Extra Credit):
    - Implement a more advanced load-balancing strategy, such as least-connections or weighted round-robin.
    - Add metrics for monitoring system performance, such as average request handling time or server load distribution.

Submission Guidelines:

1. Submit your .java file(s) and test results via Blackboard.
2. Include a short README file in Word or PDF format explaining how to run your program and any dependencies.

10-Point Rubric:

Criteria	Points	Description
Correct Load Balancing Implementation	2	Accurately implements a round-robin algorithm to distribute requests across servers.
Server Simulation	1	Simulates server behavior, including health status and failure/recovery.
Dynamic Request Reallocation	1	Detects server failures and redistributes requests to maintain availability.
User Input Handling	1	Properly handles user inputs for server count and request volume, with validations.
Output Clarity	1	Clearly displays request distribution and server status during the simulation.
Error Handling	1	Provides meaningful error messages for invalid inputs or unexpected conditions.
Documentation/Comments	1	Includes clear, concise comments explaining major code sections and logic.
Test Cases and Results	1	Provides at least three test cases demonstrating the program's functionality and resilience.
Optional Enhancements	1	Implements advanced load balancing or performance metrics (extra credit).
Overall Functionality	1	Program runs correctly, ensuring high availability as specified.
