STATE UNIVERSITY OF ZANZIBAR (SUZA)
School of Computing Communication and Media Studies (SCCMS)

Assignment 1: Multithreading & Networking

Course Code:	IT6003	**Programme:**	MSc. Information T
Course Name:	Advanced Programming Using Java	**Semester:**	I
Type:	Individual Assignment	**Total Marks:**	100

Submission: ZIP file named `RegNo_Assignment1.zip`. Include all `.java` files and a `README.txt` explaining how to compile and run each task.

-2 **Viva:** You must demonstrate your code and explain the concepts during an oral examination. Inability to explain your code results in **zero marks**.

Part A: Thread Fundamentals [25 marks]

Task 1: Thread Creation and Lifecycle [10 marks]

a) Create two classes that demonstrate both approaches to creating threads:
- `CountdownThread extends Thread`: counts down from a given number to 0
- `CountupRunnable implements Runnable`: counts up from 0 to a given number

Both should print the thread name and current count with a 500ms delay between each number. [3 marks]

b) In the `main()` method, create and start both threads. Demonstrate:
- `thread.getName()`, `thread.getId()`, `thread.getState()`
- `thread.isAlive()` before start, during execution, and after completion
- `thread.join()` to wait for both threads to finish before printing "All done"

[3 marks]

c) Set one thread as a **daemon thread**. Explain in comments what happens when the main thread exits. Run the program and observe the behavior. [2 marks]

d) Use `thread.setPriority()` to set different priorities. Print the priorities. Explain in comments why priority-based scheduling is not guaranteed. [2 marks]

Task 2: Thread Synchronization [15 marks]

Scenario: Simulate a **Bank Account** with concurrent deposits and withdrawals.

a) Create a `BankAccount` class with:

```
public class BankAccount {
    private double balance;
    private String accountNo;

    public synchronized void deposit(double amount) { ... }
    public synchronized void withdraw(double amount) { ... }
    public synchronized double getBalance() { ... }
}
```

Implement proper synchronization using the `synchronized` keyword. [3 marks]

b) Create 5 `DepositThread` instances and 5 `WithdrawThread` instances, each performing 100 transactions of random amounts (1–100). Start all 10 threads. [3 marks]

c) Use `join()` to wait for all threads. Print the final balance and verify it equals: initial balance + total deposits − total withdrawals. [2 marks]
d) **Demonstrate the race condition:** Remove the `synchronized` keyword and run again. Show that the final balance is incorrect. Explain in a comment why this happens. [3 marks]
e) Add `wait()` and `notify()` to the `withdraw()` method: if the balance is insufficient, the thread should `wait()` until a deposit is made. The `deposit()` method should call `notifyAll()` after adding funds. [4 marks]

Part B: Advanced Concurrency [25 marks]

Task 3: Producer-Consumer Problem [12 marks]

Implement the classic Producer-Consumer problem:

a) Create a `SharedBuffer` class with a fixed-size buffer (capacity 5) using an array or `LinkedList`. Implement `produce(int item)` and `consume()` methods using `wait()` and `notifyAll()`.
   - Producer waits when the buffer is full
   - Consumer waits when the buffer is empty

   [4 marks]
b) Create 3 Producer threads (each producing 10 items) and 2 Consumer threads. Print each produce/consume action with thread name and buffer size. [3 marks]
c) Re-implement the same problem using `java.util.concurrent.BlockingQueue` (`ArrayBlockingQueue` with capacity 5). Compare the code simplicity with part (a). [3 marks]
d) Use `ExecutorService` with a fixed thread pool of 5 threads instead of manually creating threads. Use `shutdown()` and `awaitTermination()`. [2 marks]

Task 4: Concurrent Collections and Atomic Operations [13 marks]

a) **Word Frequency Counter:** Read a large text file using multiple threads (divide the file into chunks). Each thread counts word frequencies in its chunk using a `ConcurrentHashMap<String, AtomicInteger>`. Merge results and display the top 20 most frequent words. [5 marks]
b) **Thread-safe Counter Comparison:** Implement a counter using three approaches:
   - `synchronized` method
   - `AtomicInteger`
   - `ReentrantLock`

   Run 10 threads, each incrementing the counter 100,000 times. Verify correctness and measure execution time for each approach using `System.nanoTime()`. Print a comparison table. [5 marks]
c) Use `CountDownLatch` to simulate a race: create 5 "runner" threads that prepare (random delay) and then wait at the starting line. When all runners are ready, the race starts simultaneously. Print finish order. [3 marks]

Part C: Networking and Sockets [30 marks]

Task 5: TCP Client-Server Chat [15 marks]

Build a simple **multi-client chat application**:

a) **Server** (`ChatServer.java`):
   - Listen on port 5000 using `ServerSocket`
   - Accept multiple client connections (each in a separate thread)
   - Maintain a list of connected clients
   - Broadcast messages from one client to all other clients
   - Handle client disconnection gracefully

   [6 marks]
b) **Client** (`ChatClient.java`):

- Connect to the server using `Socket`
- Use two threads: one for reading messages (from server), one for sending messages (from user input via `Scanner`)
- Send a username on connection
- Display messages in format: `[username]: message`
- Support a `/quit` command to disconnect

[5 marks]

c) Add the following server commands:
- `/list` — server sends back list of connected usernames
- `/whisper username message` — private message to specific user

[4 marks]

Task 6: HTTP Client and URL Processing [15 marks]

a) **URL Information Extractor:** Write a program that takes a URL as input and uses `java.net.URL` and `HttpURLConnection` to:
- Display: protocol, host, port, path, query
- Send a GET request and display: response code, content type, content length
- Read and display the first 500 characters of the response body

[5 marks]

b) **Multi-threaded File Downloader:** Create a program that downloads a file from a URL using multiple threads:
- Use `HttpURLConnection` to get the file size
- Divide the file into 4 parts
- Each thread downloads its part using the `Range` HTTP header
- Combine the parts into the final file
- Display download progress (percentage)

[6 marks]

c) **Simple HTTP Server:** Use `com.sun.net.httpserver.HttpServer` to create a basic web server that:
- Serves static HTML files from a `public/` directory
- Handles `/api/time` endpoint returning current time as JSON
- Handles `/api/echo?msg=hello` returning the query parameter
- Returns 404 for unknown paths

[4 marks]

Part D: Integration Challenge [20 marks]

Task 7: Concurrent Network Scanner [20 marks]

Build a **Network Port Scanner** that combines multithreading and networking:

a) Accept a hostname/IP address and a port range (e.g., 1–1024) as command-line arguments. [2 marks]

b) Use a thread pool (`ExecutorService` with 50 threads) to scan ports concurrently. For each port, attempt to open a `Socket` with a timeout of 200ms. [5 marks]

c) Collect results in a `ConcurrentHashMap<Integer, String>` mapping port number to status ("OPEN" or "CLOSED"). Only store open ports. [3 marks]

d) Display results sorted by port number. For well-known ports (21-FTP, 22-SSH, 25-SMTP, 53-DNS, 80-HTTP, 443-HTTPS, 3306-MySQL, 8080-Tomcat), display the service name. [3 marks]

e) Display a progress bar during scanning: `[=====> ] 50% (512/1024 ports)`. Use `AtomicInteger` to track progress across threads. [3 marks]

f) Measure and display total scan time. Compare scanning times for different thread pool sizes (10, 50, 100, 200) on the range 1–1024. Print a comparison table. [4 marks]

Grading Rubric

Part	Marks	Weight
Part A: Thread Fundamentals	25	25%
Part B: Advanced Concurrency	25	25%
Part C: Networking and Sockets	30	30%
Part D: Integration Challenge	20	20%
Total	**100**	**100%**

Viva Questions (be prepared):

1. What is the difference between `Thread` class and `Runnable` interface?
2. Explain deadlock. Can your code deadlock? How would you prevent it?
3. What is the difference between `synchronized` and `ReentrantLock`?
4. Explain the TCP three-way handshake.
5. What happens if the server crashes while a client is connected?
6. Modify your chat server to support file transfer (live coding).

End of Assignment 1