MODULE: Java Programming

ASSIGNMENT 5 – Multithreading & Synchronization

Learning Objectives

By completing this assignment, learners will:

- Understand thread creation using `Thread` and `Runnable`
- Use multiple threads to improve performance
- Apply synchronization (`synchronized` methods & blocks)
- Handle shared resources safely
- Observe race conditions and fix them
- Simulate real-world concurrent applications
- Understand thread interleaving, CPU scheduling, and data consistency

General Instructions

1. Write each question in a clean, separate `.java` file unless specified otherwise.
2. Do not use `ExecutorService` unless asked (this module teaches core threads first).
3. Use clear logging statements so the sequence of thread execution is visible.
4. Keep all shared data **private** and expose them via getters/setters.
5. Ensure proper synchronization to prevent inconsistent results.
6. Avoid sleeping threads unless required for demonstration.

Estimated Time & Difficulty

Question	Time	Difficulty
Q1	45–60 min	Intermediate
Q2	45–60 min	Intermediate → Advanced

Evaluation Rubric

Criteria	Weight
Thread correctness & behavior	40%
Use of synchronization	30%
Code structure & clarity	15%
Logging & readability	10%

Criteria	Weight
Avoiding race conditions	5%

Q1. File Downloader Simulation Using Threads

Real-World Context

Real download managers use multiple threads to download files simultaneously.
This exercise simulates that concept.

Requirements

1. Consider a list of 25 file URLs:

```
https://www.dropbox.com/photo1.jpg
https://www.dropbox.com/photo2.jpg
...
https://www.dropbox.com/photo25.jpg
```

2. Create a class `FileDownloader` that implements `Runnable`.
3. Each thread should simulate downloading a file by:
    - printing the file name
    - sleeping for a short random duration (e.g., 200–500ms)
4. Create multiple threads and assign one file to each.
5. Start all threads to simulate concurrent downloads.
6. Observe how downloads finish in non-sequential order.

Expected Output (Sample)

```
Starting download: photo1.jpg
Starting download: photo2.jpg
Starting download: photo3.jpg
Completed: photo2.jpg
Completed: photo1.jpg
Completed: photo3.jpg
...
```

Notes & Constraints

- No real downloading — just simulation.
- Use meaningful thread names for clarity.
- Do not block using large sleep times.
- No synchronization needed (each thread handles its own file).

Q2. Synchronization – Ensuring Consistent Account Balance

Real-World Context

Bank applications often deal with concurrency issues:
when multiple operations occur on the same account at the same time, data corruption can occur unless synchronization is used.

Given Class Structure

```
public class Account {
    private double balance;

    // getter setter
    public void addAmount(double amount) { ... }
    public void subtractAmount(double amount) { ... }
}
```

This account object is shared between two entities:

- **Bank** → subtracts ₹1000 **100 times**
- **Company** → deposits ₹1000 **100 times**

Without synchronization, final balance becomes inconsistent.

Requirements

Part A — Implement Bank & Company Threads

Create:

- `Bank` class implementing `Runnable`
  - Calls `subtractAmount(1000)` in a loop 100 times
- `Company` class implementing `Runnable`
  - Calls `addAmount(1000)` in a loop 100 times

Part B — Add Synchronization

Modify `Account` methods in two ways:

Option 1 (Method-level synchronization)

```
public synchronized void addAmount(double amount) { ... }

public synchronized void subtractAmount(double amount) { ... }
```

Option 2 (Block-level synchronization)

```
public void addAmount(double amount) {
    synchronized(this) { ... }
}
```

Demonstrate both in code or comments.

Part C — Run Bank & Company Threads Together

1. Create one `Account` object with an initial balance (e.g., ₹10,000).
2. Create one Bank thread and one Company thread.
3. Start both threads simultaneously.
4. Print the final balance.

Expected Behavior

Without Synchronization (Race Condition)

Output may vary each run:

```
Final Balance: 12000
Final Balance: 15000
Final Balance: 8000
```

With Synchronization (Consistent Result)

```
Final Balance: 10000
```

Notes & Best Practices

- Avoid putting long-running operations inside synchronized blocks.
- Print intermediate logs to see interleaving of threads.
- Use `Thread.currentThread().getName()` in logs.
- Demonstrate how race conditions arise without synchronization.

BONUS CHALLENGES (Optional)

⭐ Bonus 1 — Thread Pool Approach (ExecutorService)

Reimplement downloader using a fixed thread pool of size 5.
Observe faster and controlled execution.

⭐ Bonus 2 — Deadlock Demonstration

Create two locks & two threads to intentionally create a deadlock, then fix it.

⭐ Bonus 3 — Account With ReentrantLock

Replace synchronized with `ReentrantLock`
Show how `lock()` → `unlock()` works.

⭐ Bonus 4 — Atomic Variables

Modify Account to use `AtomicInteger` / `AtomicLong` and compare behavior.

Reflection Questions

1. Why is synchronization necessary when multiple threads modify shared data?
2. What happens if we forget to call `unlock()` in a `ReentrantLock` implementation?
3. Does using threads always improve performance? Why or why not?
4. How did thread interleaving affect your program's output?
5. What difference did you observe between method-level and block-level synchronization?