An-Najah National University

Course: Advanced Software Engineering

Semester: First 2025/2026

Refactored the TMPS System Running

Issue Analysis Report

Instructor: Dr. Mustafa Assaf

Prepared By

Fathi Al Heelo

12323885

Date: DEC 1,2025

First, the **class JobExecutor** has many problems.

1. Thightly coupled, dependence on ConnectionManager.
2. In the executeJob Method, it does not use conactionPool, which leads to the creation of a new connection every create an instance is created, which leads to poor performance and inefficient use of resources.
3. Also, violated OCP, SRP, and DIP.
4. And, have duplicate code in (execute jobs methods).
5. And, no abstraction.
6. No centralized way to manage templates.
7. Hard to add monitoring, permission checks, and logging.

To solve these problems:

1. Replace ConnectionManager with ConnectionPool
2. Reuse Connections instead of creating a new connection every use.
3. Applying SOLID P sach as SRP, use JobExecutionProxy to separate powers and logging, and OCP use Strategy Pattern + Factory to add job types without editing JobExecutor, also DIP depends on the interface, not a concrete class.
4. Remove duplicate code through independent strategies for each Job Type.
5. Use TemplateRegistry +JobPrototype to work clones for templetes instead of rebuilding.
6. Adding permission check + timing + logging through Proxy Pattern without modifying JobExecutor.

Connection class:

1. The problem in this class is executeQuery, because you give class behavior that may not be used.

To solve that, we can make the QueryExecutor interface, and any class can provide an implementation of the QueryExecutor interface, handling the execution of queries using a given Connection by printing a simulated execution message, cleanly separating responsibilities according to the SRP principle. Also, I apply OCP.

ConnectionManager class :

Here we have a problem: every time, it makes a new connection to the database, and this is negatively impacting performance and efficiency.

We can solve this problem by creating a connection pool, making ready-made copies, and then connecting according to the queue.

Job class: doesn't need any modification, and it's a data holder **also class user some that**.

HeavyTemplate class:

This class has a problem :
is that it creates a new Job from scratch every time, with no cloning and no abstraction, making it expensive and not scalable.

How to solve that?

By providing copy() and CreateJobInstance(), allowing templates to be cloned efficiently and jobs to be created in a unified and flexible way.

TemplateManager class:

Has many problems:
is that it rebuilds heavy templates from scratch every time, causing huge performance overhead, no reuse, duplicated code, and it's not supported by the prototype design pattern.

To fix that, create the JobPrototype class and JobTemplateRegistration class, which leads to cloning instead of rebuilding, eliminating heavy load on repeated calls. Also, remove duplicates and make the program scalable.