(For Assignment 1 — Advanced Software Development / Design Patterns)

Current Issues Report — Analysis of the Provided Codebase

(For Assignment 1 — Advanced Software Development / Design Patterns)

*This report analyzes the provided naive implementation of the **TaskMaster Processing System (TMPS)**.*
The current codebase contains numerous structural, architectural, and object-oriented problems.
Below is a detailed breakdown referencing specific classes, methods, and code symptoms.

---

1. General Architectural Issues

1.1 Lack of Modularity & High Coupling

- *Many classes are tightly coupled and depend directly on concrete implementations.*
- *Example:*
  `JobExecutor` *directly creates, uses, and closes connections through* `ConnectionManager`*, instead of depending on abstractions.*
- *The system is not extendable — adding a new job type requires modifying multiple classes.*

1.2 No Clear Separation of Concerns

Several classes mix responsibilities:

- `JobExecutor` *handles:*
  ✔ *job routing*
  ✔ *job execution logic*
  ✔ *connection acquisition*
  ✔ *permission logic (implicitly)*
  ✔ *printing/logging*

*This violates the **Single Responsibility Principle (SRP)**.*

---

2. Detailed Class-Level Issues

2.1 Connection & ConnectionManager

Issue: No Real Connection Pool

`ConnectionManager` *creates a new* `Connection` *every time:*

```
public Connection createConnection() { ... return new Connection("Conn-" + n); }
```

Problems:

- *No reuse of connections.*
- *No blocking/waiting when >10 connections.*
- *Returns* `null` *if more than 10 connections — critical error.*
- `closeConnection()` *does nothing.*

SOLID Violations

- *Violates **SRP** → responsible for creation AND lifecycle.*
- *Violates **OCP** → changing pooling behavior requires editing this class.*
- *Violates **LSP** if replaced by a real pool later.*

2.2 JobExecutor

Massive if/else Block

```
if ("EMAIL".equals(job.getType())) { ... }
else if ("DATA".equals(job.getType())) { ... }
else if ("REPORT".equals(job.getType())) { ... }
```

Problems:

- *Classic violation of **Strategy Pattern**.*
- *Violates **OCP** — adding new job type requires modifying this class.*
- *Hard-coded logic is not reusable.*

Executor Handles Too Many Responsibilities

- *Obtains a connection.*
- *Executes business logic.*
- *Logs.*
- *Validates job type.*
- *Closes connection.*

*Violates **SRP** and **SoC** (Separation of Concerns).*

Missing Error Handling

- *No try/catch around strategy execution.*
- *Could cause leaked connections.*

2.3 HeavyTemplate & TemplateManager

Rebuilding Heavy Templates Every Time

The system uses:

`simulateHeavyLoad(...)`

`return new HeavyTemplate(...)`

and then creates a job with:

`createJobInstance()`

Problems:

- *Violates **Prototype Pattern** requirement.*
- *Very wasteful → 3 seconds delay for each template.*
- *No caching, no reuse, no cloning.*
- *TemplateManager has duplicated code for each template type.*

2.4 Job Class

Job stores type as String

- *Fragile.*
- *Error-prone.*
- *No enumeration or type safety.*

Job contains configuration as a raw String

- *Makes parsing, validation, and extension difficult.*

2.5 User Class

Naive Permissions

- `hasPermission()` *checks raw strings in a list.*
- *No enum or role abstraction.*
- *No permission validation at execution level.*

2.6 Missing Proxy for Controlled Execution

Current system lacks:

- *Permission validation*
- *Logging*
- *Execution timing*
- *Connection lifecycle management*
  → all of which the Proxy pattern is expected to handle.

This results in:

- *Duplicate responsibilities*
- *Insecure job execution*
- *No central monitoring*

2.7 Missing Factory Pattern for Job Strategies

The codebase does not provide:

- *A `JobStrategy` interface*
- *Concrete strategies*
- *A strategy factory*

This leads to:

- *Tight coupling*
- *No separation of algorithms*
- *Poor scalability*

---

3. Code Smells Identified

3.1 Long Method / God Object

*`JobExecutor.executeJob()` is a **God Method**:*

- *Too many concerns.*
- *Too many branches.*
- *Hard to maintain.*

3.2 Primitive Obsession

- *Job types stored as strings.*
- *Job config stored as plain text.*
- *Permission stored as plain strings.*

3.3 Duplicated Code

TemplateManager:

- *Three methods differ only by the type of template and printed text.*

3.4 Lack of Proper Error Handling

- *No handling for null connections.*
- *No handling for template loading errors.*
- *No transactional logic.*

3.5 Naive Resource Management

- *Connections are never actually released or reused.*
- *No thread safety.*

---

4. Violations of Required Patterns

Prototype Pattern missing

Templates are recreated from scratch — not cloned.

Strategy Pattern missing

JobExecutor uses if/else instead of pluggable strategies.

Proxy Pattern missing

Execution is not protected, validated, or monitored.

Connection Pool missing

ConnectionManager is not a pool — it's just a counter.

Summary of Problems

The current implementation suffers from:

- *Poor maintainability*

- *Very high coupling*
- *No abstraction*
- *Poor performance due to heavy template creation*
- *Unsafe execution model*
- *No reuse of expensive resources*
- *No design patterns applied*
- *Violations of almost all SOLID principles*

src/edu/najah/cap/advance/assignments/assignment1/templates/JobPrototype.java

```
package edu.najah.cap.advance.assignments.assignment1.templates;


import edu.najah.cap.advance.assignments.assignment1.job.Job;


/**
 * Prototype interface for job templates.
 * Concrete templates (Email, Data, Report) will implement this.
 */
public interface JobPrototype {


    /**
     * Create a copy (clone) of this template.
     */
    JobPrototype copy();
```

```
    /**
     * Create a concrete Job instance based on this template.
     */
    Job createJobInstance();
}
```

Base Prototype ليكون HeavyTemplate

عدّل ملف `HeavyTemplate.java` بالكامل إلى الشكل التالي:

```
package edu.najah.cap.advance.assignments.assignment1.templates;

import edu.najah.cap.advance.assignments.assignment1.job.Job;

/**
 * Base heavy template that simulates an expensive object.
 * Acts as an abstract Prototype.
 */
public abstract class HeavyTemplate implements JobPrototype {

    protected final String type;
    protected final String name;
    protected final String config;
    protected final String templateBody;
```

```
protected HeavyTemplate(String type, String name, String config,
                        String templateBody) {

                this.type = type;

                this.name = name;

               this.config = config;

              this.templateBody = templateBody;

                }


                    @Override

                public Job createJobInstance() {

            String id = templateBody + " _ " + type + "-" +
                  System.currentTimeMillis();

              return new Job(id, type, name, config);

                }


                public String getType() {

                return type;

                }


                public String getName() {

                return name;

                }


                public String getConfig() {

                return config;
```

```
    }

    public String getTemplateBody() {
        return templateBody;
    }
}
```

لاحظ:

- صار `HeavyTemplate` **abstract**
- `copy()` ما فيه – كل نخلي *subclass* يحدد كيف ينسخ حاله.

Concrete Prototypes

3.1 `EmailJobTemplate`

ملف جديد:
`EmailJobTemplate.java`

```
package edu.najah.cap.advance.assignments.assignment1.templates;

/**
 * Concrete prototype for email job templates.
 */
public class EmailJobTemplate extends HeavyTemplate {

    public EmailJobTemplate(String name, String config, String
templateBody) {
```

```
        super("EMAIL", name, config, templateBody);
    }

    @Override
    public JobPrototype copy() {
        // cloning is cheap: we reuse the same templateBody/config values
        return new EmailJobTemplate(this.name, this.config, this.templateBody);
    }
}
```

3.2 DataProcessingJobTemplate

```
package edu.najah.cap.advance.assignments.assignment1.templates;

/**
 * Concrete prototype for data processing job templates.
 */
public class DataProcessingJobTemplate extends HeavyTemplate {

    public DataProcessingJobTemplate(String name, String config, String templateBody) {
        super("DATA", name, config, templateBody);
    }
```

```
    @Override
    public JobPrototype copy() {
        return new DataProcessingJobTemplate(this.name, this.config,
                this.templateBody);
    }
}
```

3.3 ReportJobTemplate

```
package edu.najah.cap.advance.assignments.assignment1.templates;

/**
 * Concrete prototype for report job templates.
 */
public class ReportJobTemplate extends HeavyTemplate {

    public ReportJobTemplate(String name, String config, String
            templateBody) {
        super("REPORT", name, config, templateBody);
    }

    @Override
    public JobPrototype copy() {
        return new ReportJobTemplate(this.name, this.config,
                this.templateBody);
    }
```

```
}
```

إنشاء *JobTemplateRegistry*

ملف جديد:

JobTemplateRegistry.java

```
package edu.najah.cap.advance.assignments.assignment1.templates;


import java.util.HashMap;

import java.util.Map;

/**

* Registry for storing and cloning job template prototypes.

*/

public class JobTemplateRegistry {


private final Map<String, JobPrototype> registry = new
HashMap<>();


public void register(String key, JobPrototype prototype) {

registry.put(key, prototype);

}


/**

* Returns a cloned prototype for the given key, or null if not
found.
```

```
*/

public JobPrototype createFrom(String key) {

  JobPrototype prototype = registry.get(key);

  if (prototype == null) {

    return null;

  }

  return prototype.copy();

 }

}
```

المفتاح `key` رح نستخدمه كـ `"TYPE:TemplateName"` مثلاً:
`"REPORT:MonthlyReport"`.

تعديل `TemplateManager` لاستعمال *Prototype + Registry*

TemplateManager.java

```
package edu.najah.cap.advance.assignments.assignment1.templates;


/*

* TemplateManager that uses Prototype pattern.

* It builds heavy templates once and then clones them from a registry.

*/

public class TemplateManager {
```

```
    private final JobTemplateRegistry registry = new
            JobTemplateRegistry();


public HeavyTemplate buildEmailJobTemplate(String templateName,
                    String config) {

        String key = buildKey("EMAIL", templateName);


        JobPrototype prototype = registry.createFrom(key);

                    if (prototype == null) {

         // First time: build heavy template and register the
                        prototype

        String templateBody = simulateHeavyLoad("EmailTemplate:"
                    + templateName);

         prototype = new EmailJobTemplate(templateName, config,
                        templateBody);

                    registry.register(key, prototype);

         System.out.println("Built Email template (heavy): " +
                        templateName);

                            } else {

           System.out.println("Cloning Email template from
              prototype: " + templateName);

                                }


            return (HeavyTemplate) prototype.copy();

                            }


public HeavyTemplate buildDataProcessingTemplate(String
            templateName, String config) {
```

```
        String key = buildKey("DATA", templateName);


        JobPrototype prototype = registry.createFrom(key);

        if (prototype == null) {

    String templateBody = simulateHeavyLoad("DataTemplate:" +
            templateName);

    prototype = new DataProcessingJobTemplate(templateName,
            config, templateBody);

    registry.register(key, prototype);

    System.out.println("Built DataProcessing template
        (heavy): " + templateName);

    } else {

    System.out.println("Cloning DataProcessing template from
        prototype: " + templateName);

    }


    return (HeavyTemplate) prototype.copy();

}


public HeavyTemplate buildReportJobTemplate(String templateName,
        String config) {

    String key = buildKey("REPORT", templateName);


    JobPrototype prototype = registry.createFrom(key);

    if (prototype == null) {

    String templateBody = simulateHeavyLoad("ReportTemplate:"
            + templateName);
```

```
            prototype = new ReportJobTemplate(templateName, config,
                    templateBody);

                registry.register(key, prototype);

            System.out.println("Built Report template (heavy): " +
                    templateName);

                        } else {

               System.out.println("Cloning Report template from
                   prototype: " + templateName);

                            }


                return (HeavyTemplate) prototype.copy();

                            }


        private String buildKey(String type, String templateName) {

                return type + ":" + templateName;

                            }


             private String simulateHeavyLoad(String msg) {

         System.out.println("Simulating heavy template creation for: "
                            + msg);

                                try {

                            Thread.sleep(3000);

                        } catch (InterruptedException e) {

                          Thread.currentThread().interrupt();

                            }

                        return "Large template";
```

```
    }
}
```

- أول مرة تنادي `buildReportJobTemplate("MonthlyReport", ...)` → يبني *template* يحفظ + ثقيل *prototype*.
- ثاني مرة تنادي نفس الاسم → ما في `Thread.sleep(3000)`، بس ***clone*** من البروتوتايب **سريع**.

وما غيّرنا `MainApp`:

```
Job reportJob = templateManager
.buildReportJobTemplate("MonthlyReport",
"format=PDF;brand=TaskMaster")
.createJobInstance();
```

إنشاء *ConnectionPool*

ملف جديد:
`ConnectionPool.java`

```
package edu.najah.cap.advance.assignments.assignment1.connections;

import java.util.LinkedList;

import java.util.Queue;

/**
* Proper Connection Pool supporting up to 10 reusable connections.
* acquire() blocks if no connection is available.
*/

public class ConnectionPool {


private final int MAX = 10;
```

```
private final Queue<Connection> available = new LinkedList<>();
private int created = 0;

public synchronized Connection acquire() {
// If there is an available connection → reuse it
if (!available.isEmpty()) {
return available.poll();
}

// If pool has not reached max → create new one
if (created < MAX) {
created++;
Connection c = new Connection("Conn-" + created);
return c;
}

// Otherwise → wait until someone releases
while (available.isEmpty()) {
try {
wait();
} catch (InterruptedException e) {
Thread.currentThread().interrupt();
}
}
```

```
        return available.poll();
    }

    public synchronized void release(Connection c) {
        available.offer(c);
        notify(); // wake one waiting thread
    }
}
```

ليصبح واجهة بسيطة للـ *ConnectionManager* *Pool*

بدل ما يكون هو المسؤول عن إنشاء *connections*، نخليه ***wrapper*** فقط.

عدّل ملف *ConnectionManager.java* إلى التالي:

```
package edu.najah.cap.advance.assignments.assignment1.connections;

/**
 * Facade over the ConnectionPool.
 * Still keeps the same method names so old code doesn't break.
 */
public class ConnectionManager {

    private final ConnectionPool pool = new ConnectionPool();
```

```
    public Connection createConnection() {
        // now this uses the pool (not naive creation)
        return pool.acquire();
    }

    public void closeConnection(Connection c) {
        pool.release(c);
    }
}
```

- **ملف جديد:**
  JobStrategy.java

```
package edu.najah.cap.advance.assignments.assignment1.executor;

import edu.najah.cap.advance.assignments.assignment1.job.Job;
import edu.najah.cap.advance.assignments.assignment1.connections.Connection;

public interface JobStrategy {
    void execute(Job job, Connection connection);
}
```

EmailJobStrategy

- **ملف جديد:**
  EmailJobStrategy.java

```
package edu.najah.cap.advance.assignments.assignment1.executor;

import edu.najah.cap.advance.assignments.assignment1.connections.Connection;
```

- ```
  import edu.najah.cap.advance.assignments.assignment1.job.Job;
  
  public class EmailJobStrategy implements JobStrategy {
  
      @Override
      public void execute(Job job, Connection connection) {
          System.out.println("[EmailJob] Preparing to send email using config: " + job.getConfig());
          connection.executeQuery("INSERT INTO email_sent (job, status) VALUES ('" + job.getId() + "', 'SENT')");
      }
  }
  ```
- ---
- # 3DataProcessingStrategy
- ```
  package edu.najah.cap.advance.assignments.assignment1.executor;
  
  import edu.najah.cap.advance.assignments.assignment1.connections.Connection;
  import edu.najah.cap.advance.assignments.assignment1.job.Job;
  
  public class DataProcessingStrategy implements JobStrategy {
  
      @Override
      public void execute(Job job, Connection connection) {
          System.out.println("[DataJob] Reading & transforming data using config: " + job.getConfig());
          connection.executeQuery("SELECT * FROM source_table WHERE job_id = '" + job.getId() + "'");
          connection.executeQuery("INSERT INTO processed_results (job_id) VALUES ('" + job.getId() + "')");
      }
  }
  ```
- ---
- # 4ReportGenerationStrategy

```
package edu.najah.cap.advance.assignments.assignment1.executor;

import edu.najah.cap.advance.assignments.assignment1.connections.Connection;
import edu.najah.cap.advance.assignments.assignment1.job.Job;

public class ReportGenerationStrategy implements JobStrategy {

    @Override
    public void execute(Job job, Connection connection) {
        System.out.println("[ReportJob] Generating report (" + job.getName() + ") using config: " + job.getConfig());
        connection.executeQuery("SELECT * FROM report_source WHERE report = '" + job.getName() + "'");
        connection.executeQuery("INSERT INTO generated_reports (job_id, path) VALUES ('" +
                job.getId() + "', '/reports/" + job.getId() + ".pdf')");
    }
}
```

---

5 JobStrategyFactory

ملف جديد:
JobStrategyFactory.java

```
package edu.najah.cap.advance.assignments.assignment1.executor;

public class JobStrategyFactory {

    public static JobStrategy getStrategy(String jobType) {

        if (jobType == null) return null;

        switch (jobType.toUpperCase()) {
            case "EMAIL":
                return new EmailJobStrategy();
            case "DATA":
                return new DataProcessingStrategy();
            case "REPORT":
                return new ReportGenerationStrategy();
```

```
            default:
                return null;
        }
    }
}
```

---

6 Refactor JobExecutor to Use Strategy

ا
JobExecutor.java
بهذا الكود:

```
package edu.najah.cap.advance.assignments.assignment1.executor;

import edu.najah.cap.advance.assignments.assignment1.connections.Connection;
import edu.najah.cap.advance.assignments.assignment1.connections.ConnectionManager;
import edu.najah.cap.advance.assignments.assignment1.job.Job;

/**
 * Refactored JobExecutor using Strategy Pattern.
 */
public class JobExecutor {

    private final ConnectionManager cm;

    public JobExecutor(ConnectionManager cm) {
        this.cm = cm;
    }

    public void executeJob(Job job) {
        System.out.printf("[Executor] Starting job %s (%s) requested by %s%n",
                job.getName(), job.getType(),
                job.getRequestedBy() == null ? "unknown" : job.getRequestedBy().getName());

```

```
        JobStrategy strategy = JobStrategyFactory.getStrategy(job.getType());

        if (strategy == null) {
            System.out.println("[Executor] Unknown job type: " + job.getType());
            return;
        }

        Connection connection = cm.createConnection();

        try {
            strategy.execute(job, connection);
        } finally {
            cm.closeConnection(connection);
            System.out.printf("[Executor] Finished job %s%n", job.getName());
        }
    }
}
```

1Add Permission Model (Optional but Required by Proxy)

نعدّل الـ *User class* بحيث يكون *permission* واضح.

لكن **ما بدنا نغيّر الكلاس نفسه** — نستخدمه زي ما هو.
Proxy رح يسأل:

- `job.getRequestedBy().hasPermission(job.getType())`

2Create JobExecutorProxy

ملف جديد:
JobExecutorProxy.java

```
package edu.najah.cap.advance.assignments.assignment1.executor;

import edu.najah.cap.advance.assignments.assignment1.connections.Connection;
import edu.najah.cap.advance.assignments.assignment1.connections.ConnectionManager;
import edu.najah.cap.advance.assignments.assignment1.job.Job;
import edu.najah.cap.advance.assignments.assignment1.model.User;

/**
 * Proxy that controls job execution:
 * - validates permissions
 * - logs start/end
 * - measures execution time
 * - acquires and releases DB connections
 */
public class JobExecutorProxy {

    private final JobExecutor realExecutor;
    private final ConnectionManager connectionManager;

    public JobExecutorProxy(JobExecutor executor, ConnectionManager cm) {
        this.realExecutor = executor;
        this.connectionManager = cm;
    }

    public void execute(Job job) {

        User user = job.getRequestedBy();
        if (user == null) {
            System.out.println("[Proxy] ERROR: Job has no requesting user.");
            return;
        }

        String jobType = job.getType();
        if (!user.hasPermission(jobType)) {
```

```
            System.out.println("[Proxy] ERROR: User " +
user.getName() +
                    " does NOT have permission to execute job
type: " + jobType);
            return;
        }

        System.out.printf("[Proxy] User %s starting job %s
(%s)%n",
                user.getName(), job.getName(), jobType);

        long start = System.currentTimeMillis();

        Connection c = connectionManager.createConnection();

        try {
            // Delegate to real executor logic
            realExecutor.executeWithConnection(job, c);
        } finally {
            connectionManager.closeConnection(c);

            long end = System.currentTimeMillis();
            System.out.printf("[Proxy] Job %s finished in %d
ms%n",
                    job.getName(), (end - start));
        }
    }
}
```

---

3 Modify Real JobExecutor (clean version)

لم يعد مسؤول عن *JobExecutor*:

- *acquiring / releasing connections*

- *permissions*
- *logging*
- *timing*

جديدة اسمها ***method*** **نضيف فقط**:

```
executeWithConnection(job, connection)
```

والـ *Proxy* رح يستدعيها.

لذلك:

استبدل محتوى *JobExecutor.java* بهذا:

```
package edu.najah.cap.advance.assignments.assignment1.executor;

import edu.najah.cap.advance.assignments.assignment1.connections.Connection;
import edu.najah.cap.advance.assignments.assignment1.job.Job;

/**
 * Clean JobExecutor after refactoring:
 * - No permission checks
 * - No connection handling
 * - No logging
 * Only responsible for running strategy.
 */
public class JobExecutor {

    public void executeWithConnection(Job job, Connection connection) {

        JobStrategy strategy = JobStrategyFactory.getStrategy(job.getType());

        if (strategy == null) {
            System.out.println("[Executor] Unknown job type: " + job.getType());
            return;
        }

        strategy.execute(job, connection);
```

- `        }`
- `}`

صار *JobExecutor* فعلاً *"single-responsibility"* و *strategy-based.*

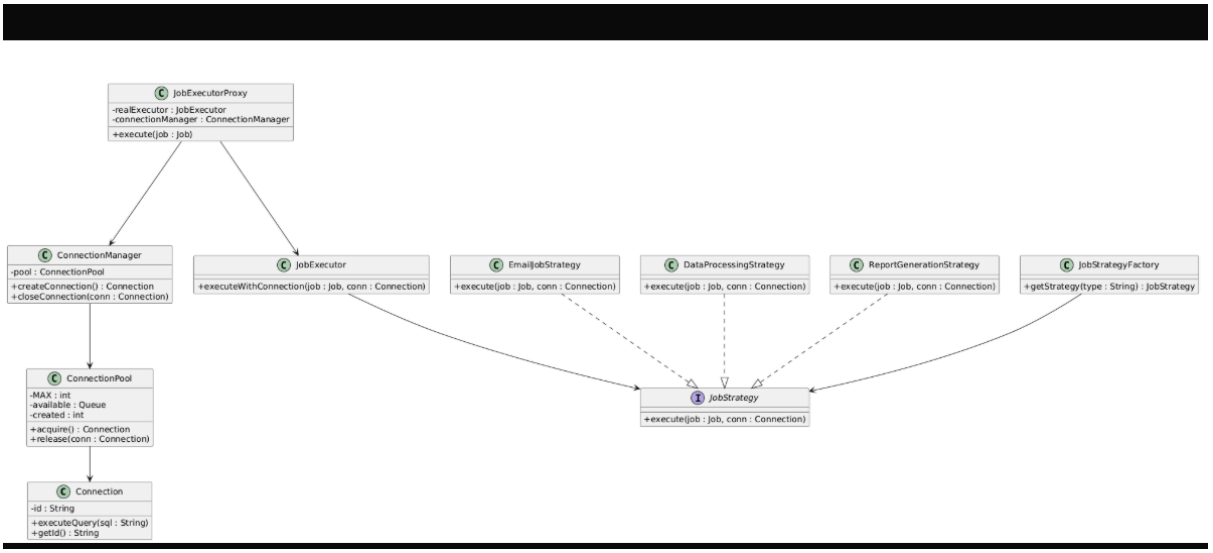
4Update MainApp to use Proxy

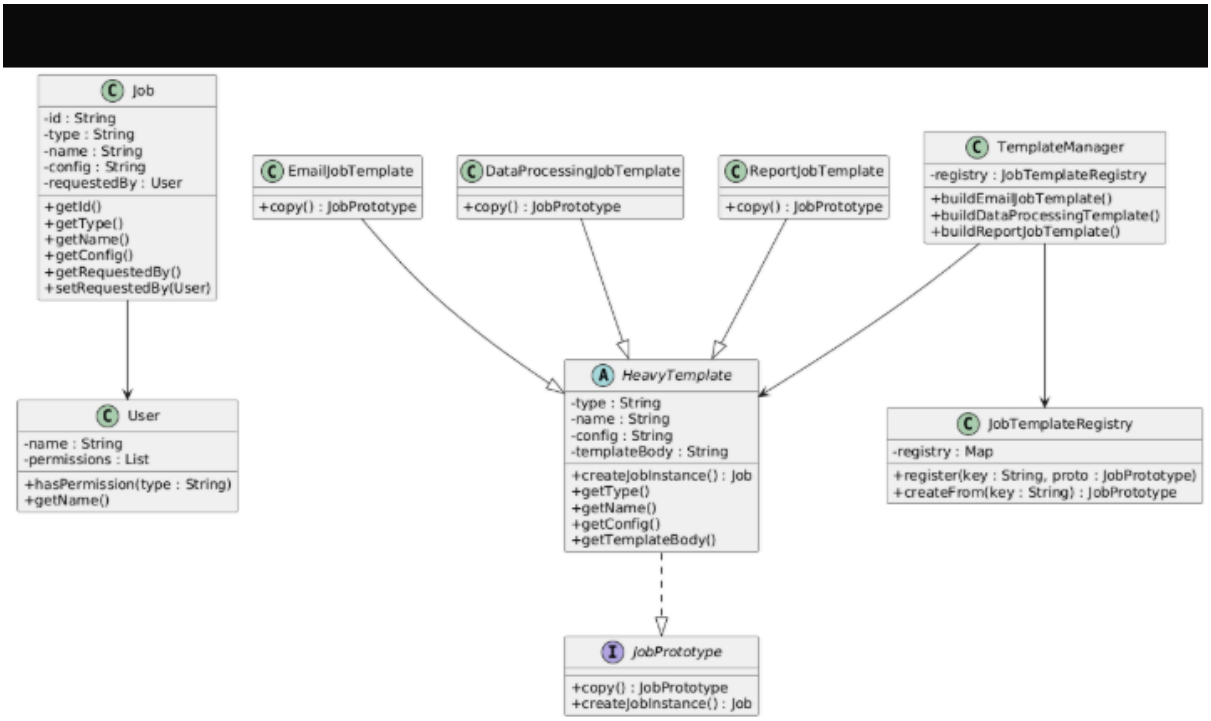
- `ConnectionManager connManager = new ConnectionManager();`
-
- `// real executor (strategy only)`
- `JobExecutor realExecutor = new JobExecutor();`
-
- `// proxy (permissions + logging + connection pool)`
- `JobExecutorProxy proxy = new JobExecutorProxy(realExecutor, connManager);`

صارت عنا

إلى:

```
proxy.execute(reportJob);
```




Job
-id : String
-type : String
-name : String
-config : String
-requestedBy : User
+getId()
+getType()
+getName()
+getConfig()
+getRequestedBy()
+setRequestedBy(User)
EmailJobTemplate
+copy() : JobPrototype
DataProcessingJobTemplate
+copy() : JobPrototype
ReportJobTemplate
+copy() : JobPrototype
TemplateManager
-registry : JobTemplateRegistry
+buildEmailJobTemplate()
+buildDataProcessingTemplate()
+buildReportJobTemplate()
User
-name : String
-permissions : List
+hasPermission(type : String)
+getName()
HeavyTemplate
-type : String
-name : String
-config : String
-templateBody : String
+createJobInstance() : Job
+getType()
+getName()
+getConfig()
+getTemplateBody()
JobTemplateRegistry
-registry : Map
+register(key : String, proto : JobPrototype)
+createFrom(key : String) : JobPrototype
JobPrototype
+copy() : JobPrototype
+createJobInstance() : Job