# Job Management System

## Table of Contents



## Definitions

**JMS** - Job Management System
**DWH** - Data Warehouse

## Assumptions

**Single Application**
**Asynchronous Job Execution**
**Scheduled Time > Priority** - Priority only taken into account on same schedules
**Using Java SE-15**

# I. Overview

The goal of this system is to handle the execution of multiple types of Jobs. The actions performed by these Jobs are not important; possible examples of these Jobs could be performing a data-load into a DWH, performing indexing of some file based content or sending emails.
Features of this system are:

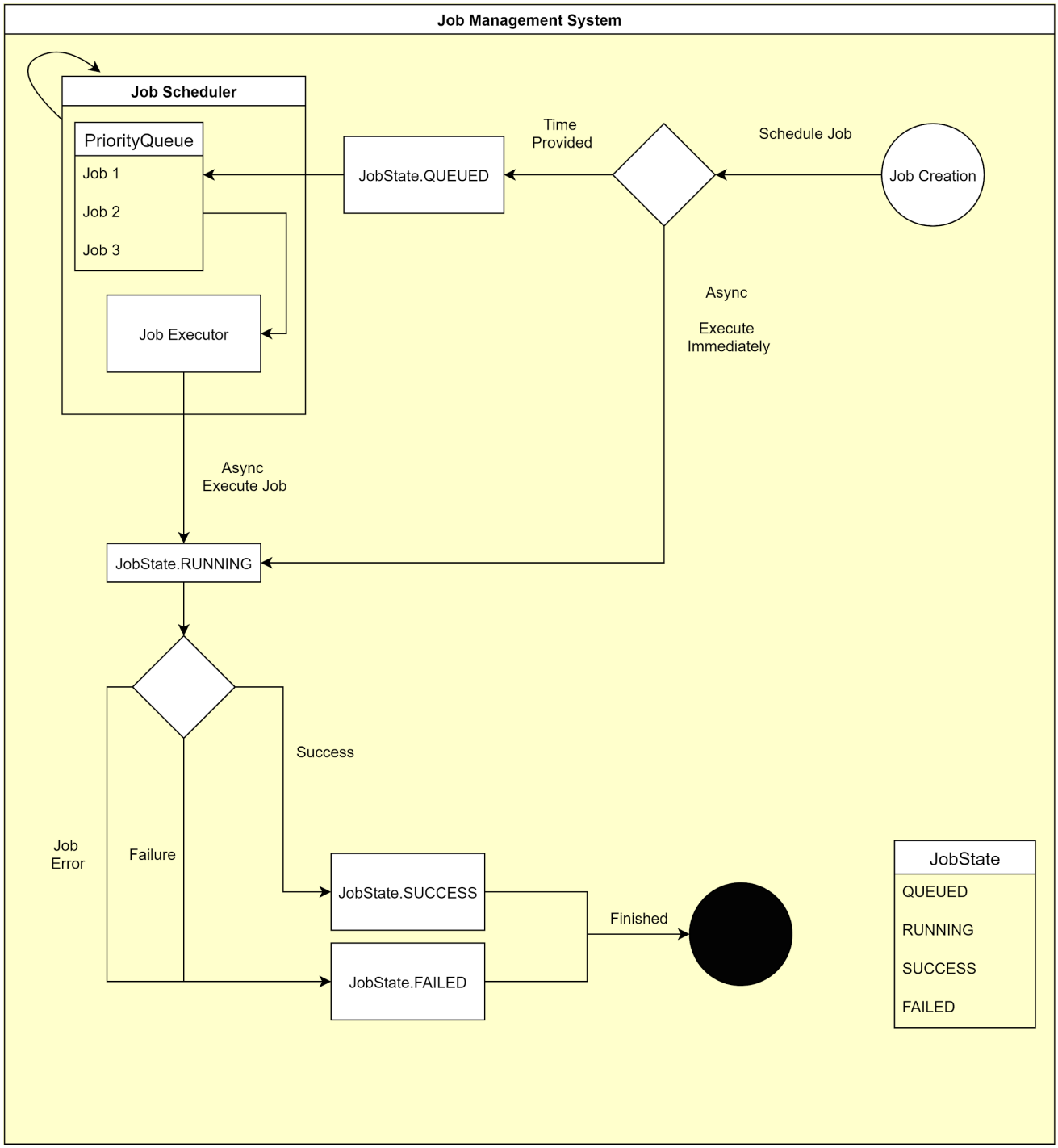
- **Flexibility**
  - The types of possible actions performed by the Jobs are not known to the Job Management System. In the future, new Jobs should be supported without re-developing the Job Management System (optional).
- **Reliability**
  - Each Job should either complete successfully or perform no action at all. (I.e. there should be no side-effects created by a Job that fails.)
- **Internal Consistency**
  - At any one time a Job has one of four states: QUEUED, RUNNING, SUCCESS, FAILED. Following the execution of a Job, it should be left in an appropriate state.
- **Priority**
  - Each Job can be executed based on its priority relative to other Jobs
- **Scheduling**
  - A Job can be executed immediately or according to a schedule

Achieving each requirement:

- **Flexibility**
    - To achieve flexibility in the creation of new jobs to be supported, all created jobs will extend an abstraction of what a job should be. All jobs have one purpose which is to execute an action. Through the use of the command pattern this is easily achieved. A job will only know one purpose which is to execute and not need to know the details.
- **Reliability**
    - To ensure reliability, job execution will be handled by an executor. The executor will execute the jobs and handle errors/exceptions accordingly when job execution occurs.
- **Internal Consistency**
    - To ensure internal consistency, the job will consider four states of operation: QUEUED, RUNNING, SUCCESS, FAILED.
    - A job state will be set to QUEUED when the job is scheduled by the scheduler., upon job execution, the job executor will be in charge of the three states - RUNNING,SUCCESS, and FAILED. Upon execution by the job executor, the job state will be set to RUNNING. The executor will handle the return response of the job and set the state accordingly to SUCCESS or FAILED. If an error is to occur during job execution the executor will catch accordingly and set the state of the job to FAILED.
- **Priority**
    - Jobs can be given a priority upon creation. Priority is taken into account by the scheduler when jobs are scheduled during the same time. The job with the higher priority will be scheduled first.
- **Scheduling**
    - The JMS uses the scheduler to perform all operations of a Job. A job added to the scheduler will be prioritized based on the time scheduled. A job will be executed immediately if scheduled with no time provided. In order to ensure a job does not block other jobs upon its execution, the jobs are executed asynchronously.

# II. Design

- JobManagementService
    - The purpose of the JobManagementService is to act as the main component when running the system. The service will be in charge of starting the scheduler to allow jobs to be scheduled. Through the service jobs can be scheduled and executed accordingly.
- JobScheduler
    - The purpose of the JobScheduler is to allow multiple types of jobs to be scheduled based on the given time provided. When a job is scheduled it sets the state of the job to QUEUED. The scheduler runs asynchronously and continuously checks to determine if a job scheduled is ready for execution. When a job is ready to be executed, it sends the job to the executor.
- JobExecutor
    - The purpose of the JobExecutor is to handle job executions accordingly. The executor will ensure the state of the job is correctly handled. When a job is executed the executor will set the state of the job to RUNNING. Upon its completion the executor will set the state accordingly to result and handle any exceptions that the job may produce. A job returns a code that can be used to identify the success or failure of the job. Codes <0-699 are reserved for error related codes, 700+ are reserved for success codes.
- Job
    - The purpose of Job is to allow flexibility in the creation of multiple types of jobs to ensure the system can run a job without knowing the type of job executing. To achieve this the job is an abstraction for other types of jobs and makes use of the command pattern for execution.
        - To showcase this, two jobs were created from abstraction - CommandJob and WaitForFileEvent
            - CommandJob - Generic Job that uses the command pattern to be able to execute any type of command. This job can be implemented to execute any action such as printing Hello World to reading and processing a file,etc.
            - WaitForFileEvent - This job does exactly how it sounds, given a filename, it will wait for the provided file or until it will timeout.


Job Management System
Job Scheduler
PriorityQueue
Job 1
Job 2
Job 3
Job Executor
JobState.QUEUED
Time Provided
Schedule Job
Job Creation
Async
Execute Immediately
Async Execute Job
JobState.RUNNING
Success
Job Error
Failure
JobState.SUCCESS
JobState.FAILED
Finished
JobState
QUEUED
RUNNING
SUCCESS
FAILED

# III. Next Steps Overview

This version of the JMS is a very simple prototype. It was created assuming only single application functionality.

- Time
    - Currently the scheduler schedules jobs based on a given epoch in milliseconds. This can get quite confusing. Improvements to be made to allow for inputting a Date/Time rather than milliseconds. In terms of a simple prototype, milliseconds are being used.
- Client-Side JMS
    - Currently, the JMS only runs and tests within the JMS application itself. Next steps would be building out the client side JMS to allow creation and scheduling of jobs to be sent to the JMS to process. Changes to the JMS will be needed as well to act as a server and respond accordingly to client requests.
- Database
    - The scheduler currently uses a simple priority queue assuming jobs are created and executed on a one time basis. This implementation makes it so that once a job is created and scheduled, the details can no longer be changed to affect its position within the queue unless the job is removed and rescheduled. Next steps would be the use of a **database**.
    - A database will allow for improvements on a large scale. Client JMS can create jobs and the database can retain the creation of these jobs to allow for the scheduler to schedule the jobs on a recurring basis,unschedule the job, or edit the job schedule.
    - The client can edit the job accordingly without having to create a whole new job and the database would be able to retain the changes.
    - The database will allow for more reliability. The scheduler can maintain the schedule it has by keeping track via the database. In case the JMS were to shutdown in certain scenarios, data will be saved and be able to restart upon the JMS starting back up.
- UI
    - Creation of a UI for the client side JMS to allow **visualization.**
        - Jobs being executed, the state of the job, and more.
        - Creation and editing of jobs.
        - Scheduling jobs.
        - Viewing how long the jobs have been running/runtime of the jobs.

- Workflows
    - A workflow will be a type of runnable to batch multiple jobs together. The workflow will work in such a way that it has a start and end. Jobs will be specified to run in an order and will only run based on conditions of the previous job in the workflow. For example: Job1 -> Job2 -> Job3. Job1 might be waiting for a file to to appear while Job2 will then perform some operations on the file, and Job3 will then process the file. Job2 however won't execute until Job1 completion.