Spring Scheduler

🌱 What is Spring Scheduler?

Spring Scheduler is a part of the Spring Framework that allows you to **schedule tasks** to run at specific intervals or times using:

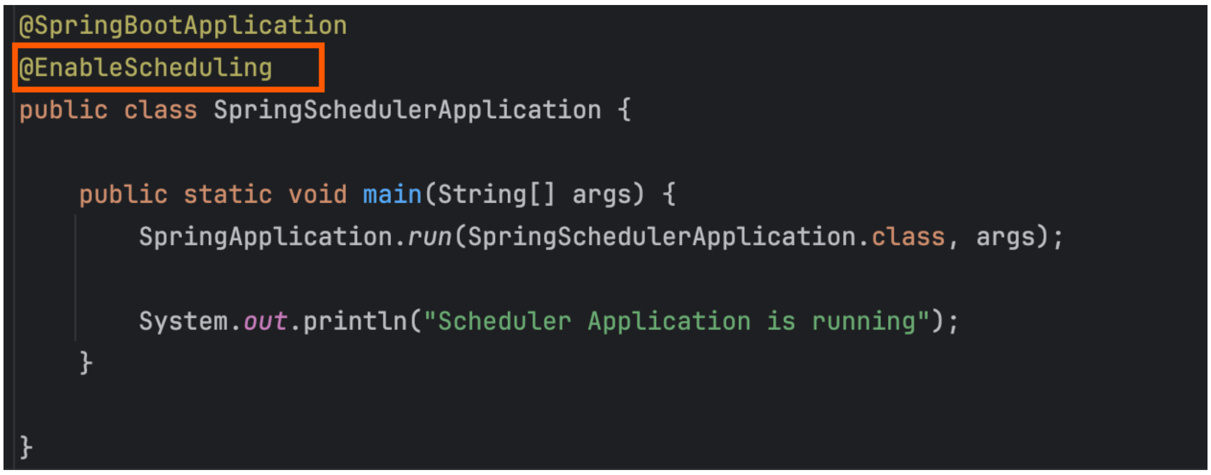
- **Fixed rate** (e.g., every 5 seconds)
- **Fixed delay** (e.g., 5 seconds after previous execution)
- **Cron expression** (e.g., every day at 12:00 PM)

it uses @Scheduled annotation and works well for background jobs or periodic tasks like:

- Sending emails
- Cleaning temp files
- Syncing data from another service

⚙️ Enable Scheduling in Spring Boot

Important Add @EnableScheduling in your main class:



```
@SpringBootApplication
@EnableScheduling
public class SpringSchedulerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringSchedulerApplication.class, args);

        System.out.println("Scheduler Application is running");
    }

}
```

🔷Types of Scheduling

1.Cron Expression: A **cron expression** is a string representing a schedule using **6 fields** (Spring) or **7 fields** (Quartz). It defines when a task should run.

🔷 Spring Cron Format (6 Fields)

```
┌────────────── second (0–59)
│ ┌──────────────── minute (0–59)
│ │ ┌──────────────── hour (0–23)
│ │ │ ┌──────────────── day of month (1–31)
│ │ │ │ ┌──────────────── month (1–12 or JAN–DEC)
│ │ │ │ │ ┌──────────────── day of week (0–6 or SUN–
SAT, 0=Sunday)
│ │ │ │ │ │
* * * * * *
```

🔷 Quartz Cron Format (7 Fields)

```
┌────────────── second (0–59)
│ ┌──────────────── minute (0–59)
│ │ ┌──────────────── hour (0–23)
│ │ │ ┌──────────────── day of month (1–31)
│ │ │ │ ┌──────────────── month (1–12)
│ │ │ │ │ ┌──────────────── day of week (1–7)
(1=Sunday)
│ │ │ │ │ │ ┌──────────────── year [optional]
│ │ │ │ │ │ │
* * * * * * *
```

🔹 Symbols in Cron

Symbol	Meaning
*	Any value
?	No specific value (Quartz only)
-	Range of values (1-5)
,	List of values (MON,WED,FRI)
/	Step values (0/5 → every 5 sec)
L	Last (e.g., last day of month)
W	Nearest weekday (Quartz)
#	Nth day of month (e.g., MON#3)

🔹 Common Cron Expression Examples

Cron Expression	Meaning
0 * * * * *	Every minute at 0 seconds
*/10 * * * * *	Every 10 seconds
0 */5 * * * *	Every 5 minutes
0 0 * * * *	Every hour
0 0 9 * * *	Every day at 9:00 AM
0 0 9 * * MON	Every Monday at 9:00 AM
0 15 10 ? * *	Every day at 10:15 AM (Quartz style)
0 0 0 1 * *	First day of every month at midnight
0 0 12 * * ? 2025	Every day at noon in 2025 only
0 0 12 15 6 ?	Every June 15 at 12:00 PM (Quartz)

🔹 Tools to Generate and Test Cron

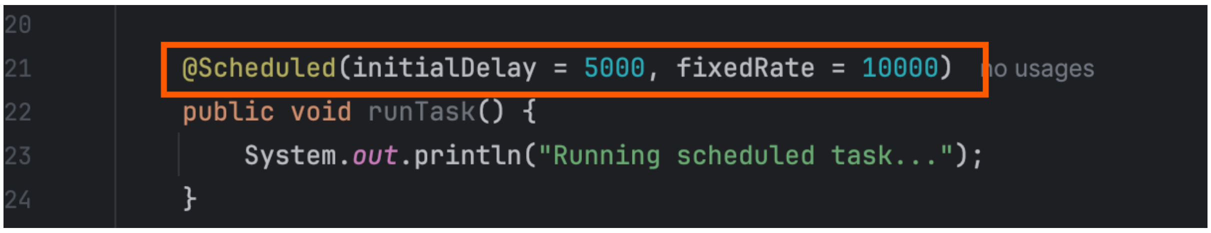
- https://crontab.guru/ – (for UNIX-style, 5-field cron)

- https://www.freeformatter.com/cron-expression-generator-quartz.html – (for Quartz)

2. initialDelay in Spring Scheduler:

initialDelay tells Spring **how long to wait before running the scheduled method for the first time**. After that, it follows the normal fixedRate or fixedDelay schedule.

Example:



```
@Scheduled(initialDelay = 5000, fixedRate = 10000)  no usages
public void runTask() {
    System.out.println("Running scheduled task...");
}
```

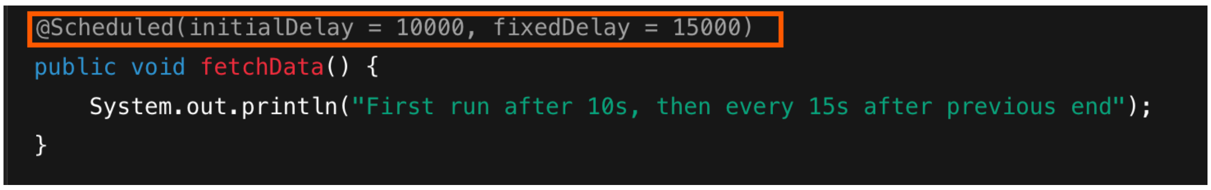
Explanation:

- initialDelay = 5000: Wait **5 seconds** (5000 ms) before the first run.
- fixedRate = 10000: Then run the method **every 10 seconds**.

🔷 Why is initialDelay useful?

- Wait for system startup →(Prevent task from running immediately before app fully starts)
- Delay for dependent services→ (Wait for external services (like DB, APIs) to be ready)
- Control first run timing→(Run task after a buffer time)

More Examples:

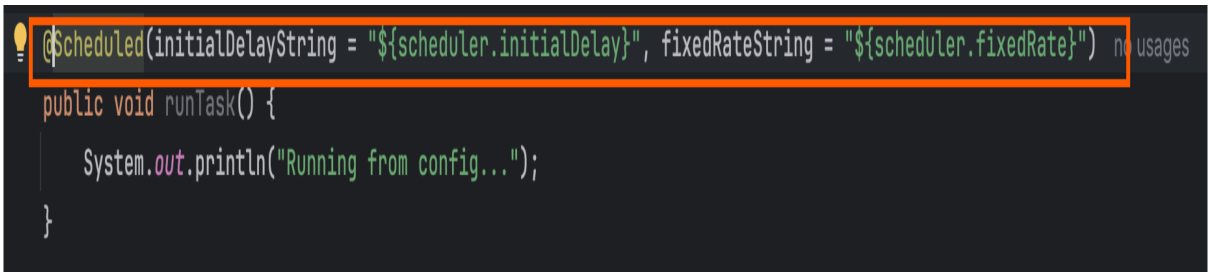


```
@Scheduled(initialDelay = 10000, fixedDelay = 15000)
public void fetchData() {
    System.out.println("First run after 10s, then every 15s after previous end");
}
```

- fixedDelay waits **after method ends**.
- fixedRate ignores execution time and runs every N ms.

Use in application.yml: If you want to move delays to config:

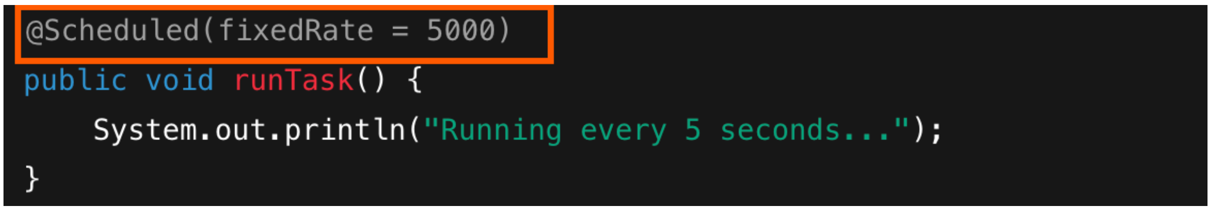
```yaml
scheduler:
  initialDelay: 5000
  fixedRate: 10000
```



```java
@Scheduled(initialDelayString = "${scheduler.initialDelay}", fixedRateString = "${scheduler.fixedRate}") no usages
public void runTask() {
    System.out.println("Running from config...");
}
```

3. @SCHEDULED(FIXEDRATE = ...) THE FIXEDRATE ATTRIBUTE IN SPRING'S @SCHEDULED ANNOTATION TELLS SPRING TO **RUN THE METHOD AT REGULAR INTERVALS**, MEASURED **FROM THE START TIME OF THE PREVIOUS EXECUTION.**

Example:

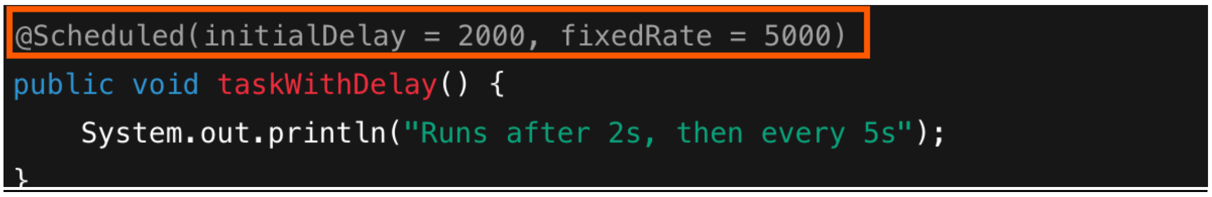


```java
@Scheduled(fixedRate = 5000)
public void runTask() {
    System.out.println("Running every 5 seconds...");
}
```

 First run: Immediately (unless initialDelay is set)

 Then: Every **5 seconds**, no matter how long the method takes

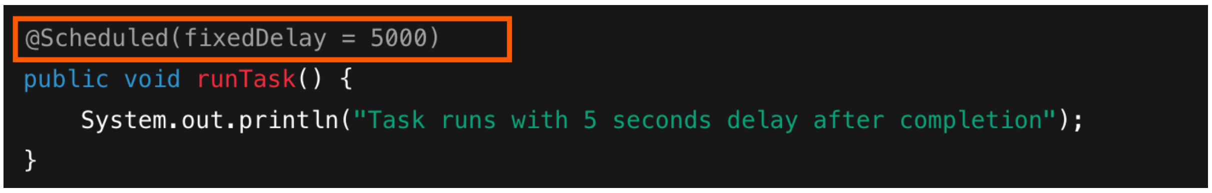
🔶 **With initialDelay**



```
@Scheduled(initialDelay = 2000, fixedRate = 5000)
public void taskWithDelay() {
    System.out.println("Runs after 2s, then every 5s");
}
```

4. @SCHEDULED(FIXEDDELAY = ...) THE FIXEDDELAY ATTRIBUTE IN SPRING'S @SCHEDULED ANNOTATION TELLS SPRING TO **WAIT FOR THE CURRENT TASK TO COMPLETE**, AND THEN WAIT FOR THE SPECIFIED DELAY **BEFORE STARTING THE NEXT EXECUTION.**

Example:



```
@Scheduled(fixedDelay = 5000)
public void runTask() {
    System.out.println("Task runs with 5 seconds delay after completion");
}
```

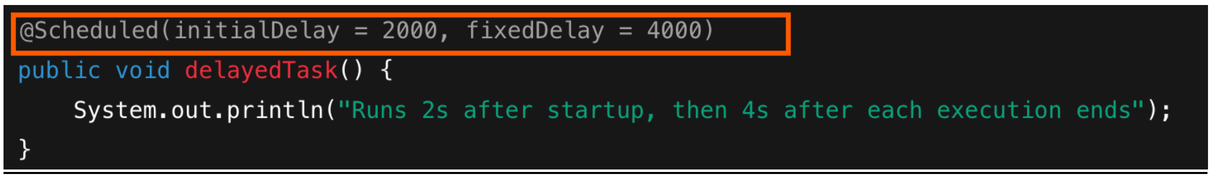
 First run: Immediately (unless initialDelay is used)

 Then: After **5 seconds** of **finishing the last run.**

Characteristics:

- Measured From-→ **End time** of the previous execution.
- Waits for completion? → **Yes**, always waits for the task to finish.
- Risk-→ Longer tasks will increase interval between runs

With initialDelay:



```
@Scheduled(initialDelay = 2000, fixedDelay = 4000)
public void delayedTask() {
    System.out.println("Runs 2s after startup, then 4s after each execution ends");
}
```

- Runs 2s after startup, then 4s after each execution ends.

*****