Spring Boot

Scheduling

Rohan Thapa
thaparohan2019@gmail.com

What is Spring Boot Scheduling?

Spring Boot Scheduling is a feature that allows you to **execute tasks** at **fixed intervals** or **specific times**, **without human intervention**.

It leverages **Spring's @Scheduled** annotation to run methods at predefined intervals.

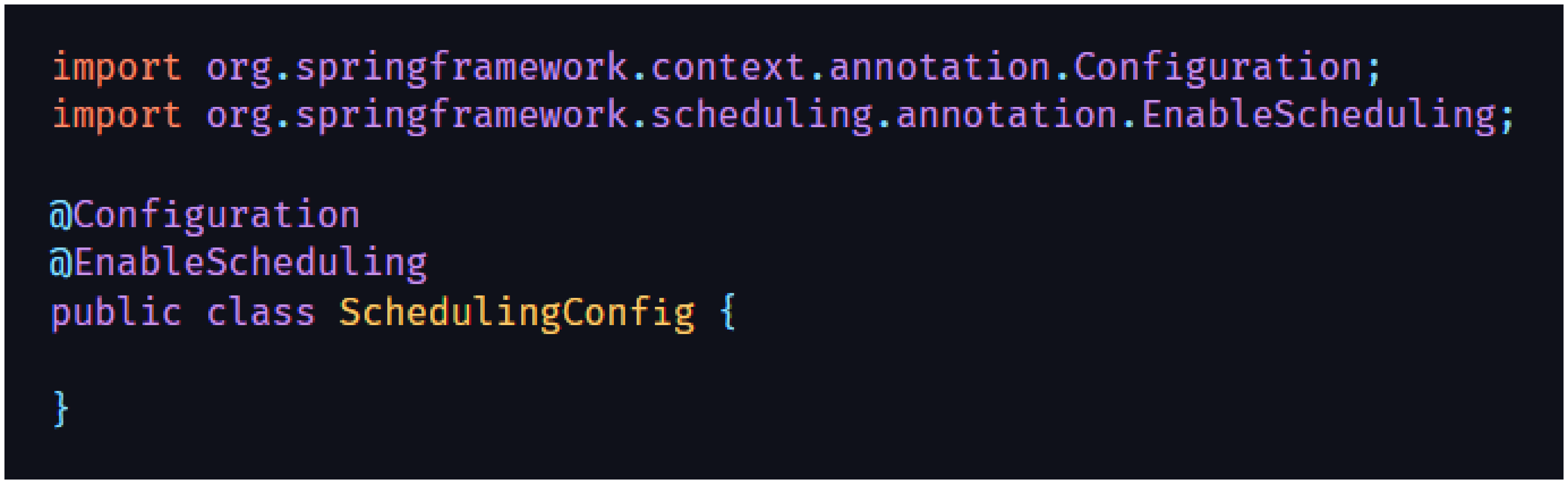
Why Use Scheduling?

Scheduling is useful for **automating** tasks that need to run periodically, such as:

- Generating reports.
- Cleaning up resources (e.g., deleting old files or data).
- Sending emails or notifications.
- Polling services or APIs.

Enabling Scheduling in Spring Boot

To enable scheduling, you need to add the **@EnableScheduling** annotation to a configuration class:



```java
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;

@Configuration
@EnableScheduling
public class SchedulingConfig {

}
```

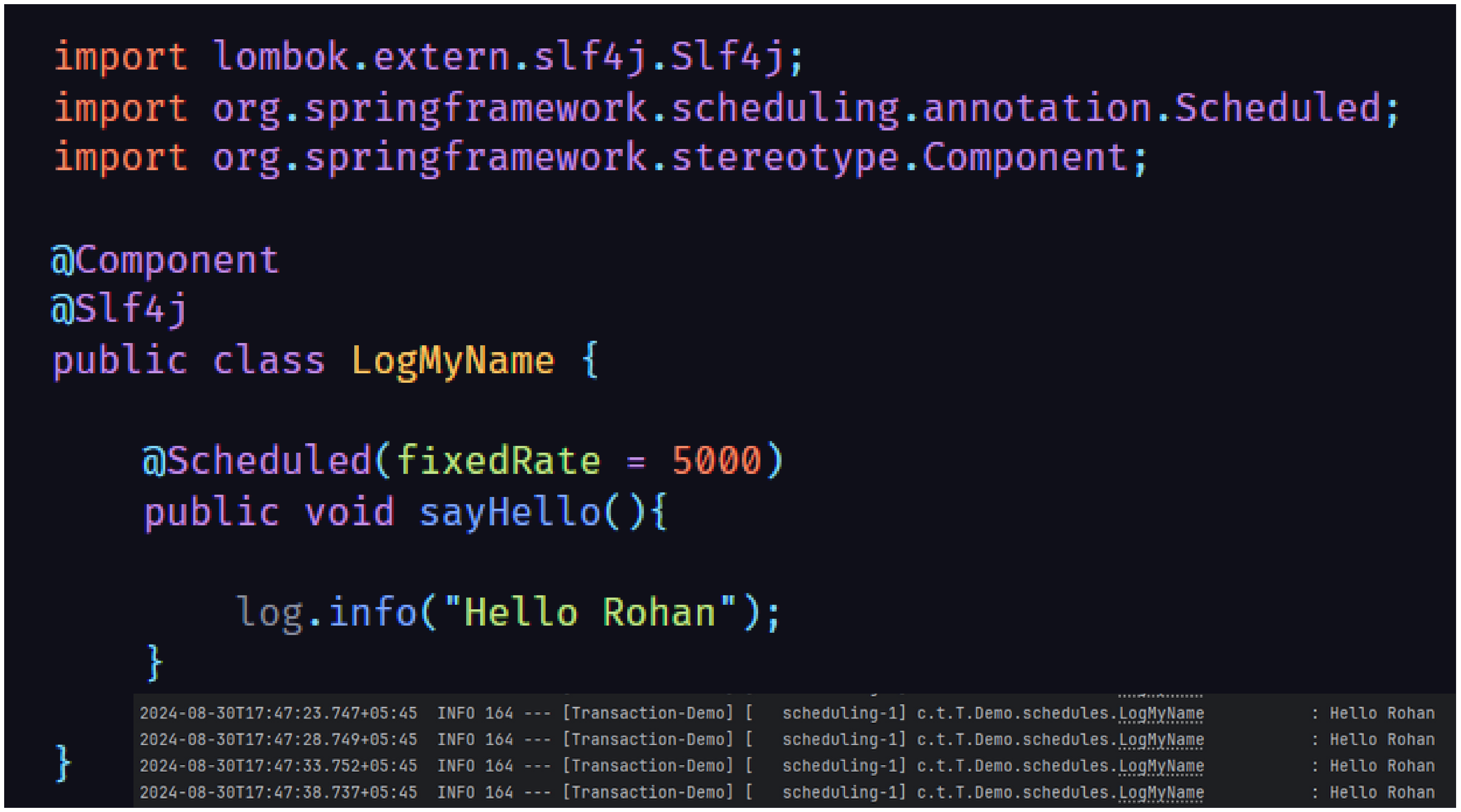
Using the @Scheduled Annotation

The **@Scheduled** annotation is used to mark methods that should be executed on a schedule. It supports various scheduling options:

1. Fixed Rate Execution
2. Fixed Delay Execution
3. Cron Expression

Fixed Rate Execution

Runs the method at a fixed interval, measured from the start of each execution.



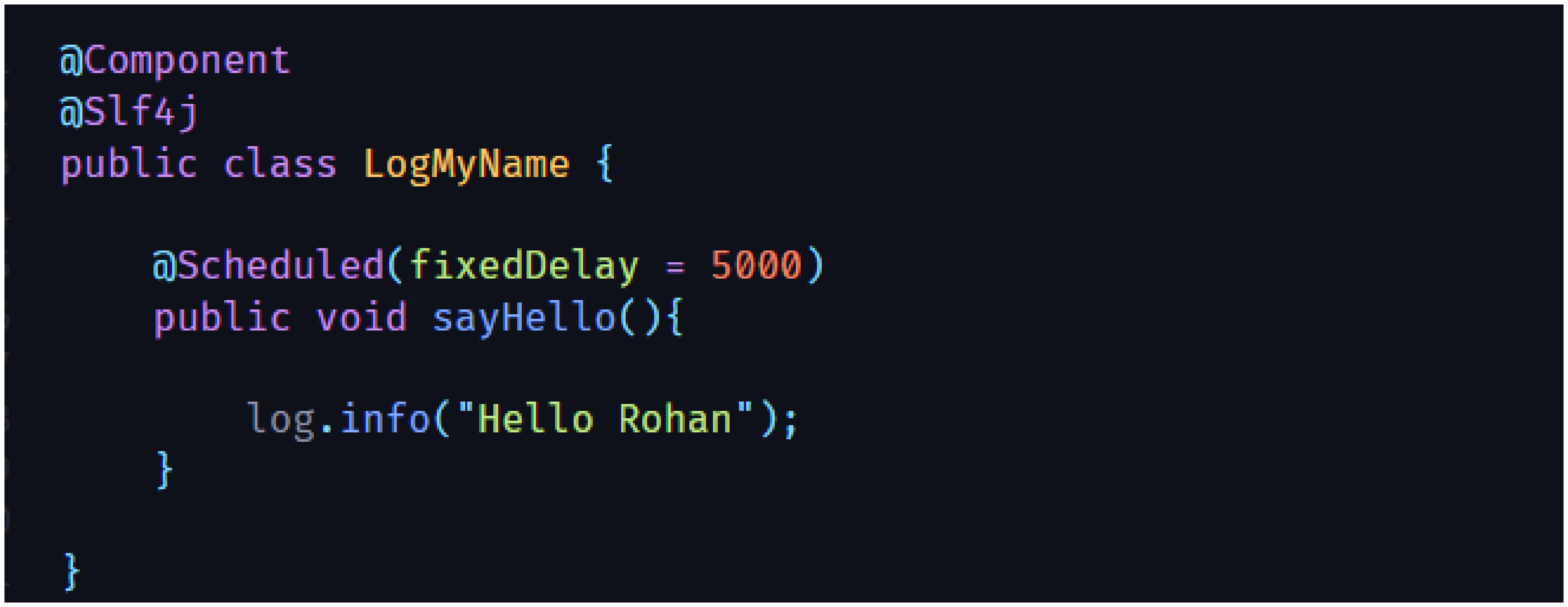
```java
import lombok.extern.slf4j.Slf4j;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
@Slf4j
public class LogMyName {

    @Scheduled(fixedRate = 5000)
    public void sayHello(){

        log.info("Hello Rohan");
    }

}
```

```
2024-08-30T17:47:23.747+05:45  INFO 164 --- [Transaction-Demo] [   scheduling-1] c.t.T.Demo.schedules.LogMyName           : Hello Rohan
2024-08-30T17:47:28.749+05:45  INFO 164 --- [Transaction-Demo] [   scheduling-1] c.t.T.Demo.schedules.LogMyName           : Hello Rohan
2024-08-30T17:47:33.752+05:45  INFO 164 --- [Transaction-Demo] [   scheduling-1] c.t.T.Demo.schedules.LogMyName           : Hello Rohan
2024-08-30T17:47:38.737+05:45  INFO 164 --- [Transaction-Demo] [   scheduling-1] c.t.T.Demo.schedules.LogMyName           : Hello Rohan
```

This example runs the task every 5 seconds, regardless of how long the task takes to complete.

Fixed Delay Execution

Runs the method after a fixed delay, measured from the completion of the last execution.

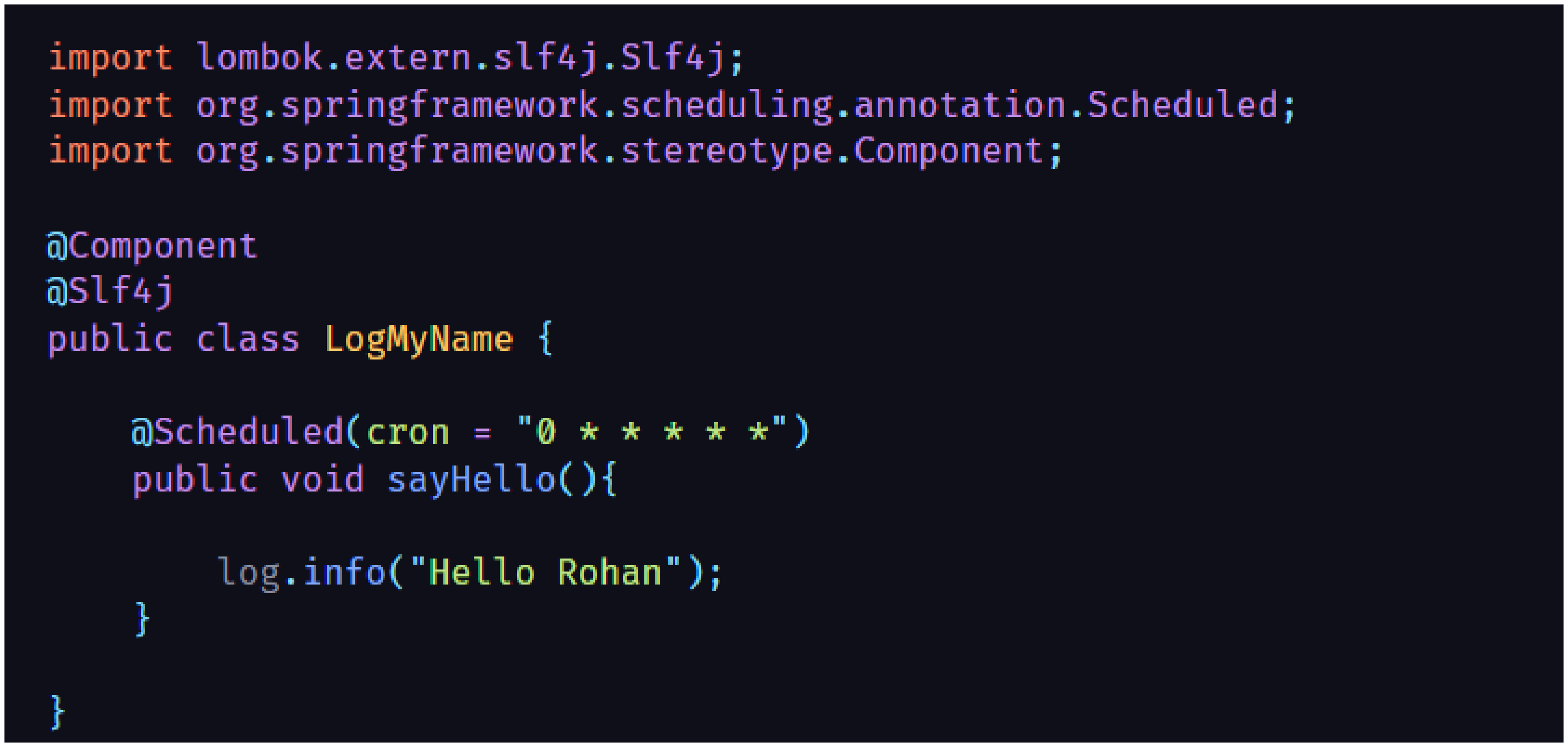


```java
@Component
@Slf4j
public class LogMyName {

    @Scheduled(fixedDelay = 5000)
    public void sayHello(){

        log.info("Hello Rohan");
    }

}
```

This example ensures that there’s a 5-second delay after the last execution finishes before the next execution starts.

Cron Expression

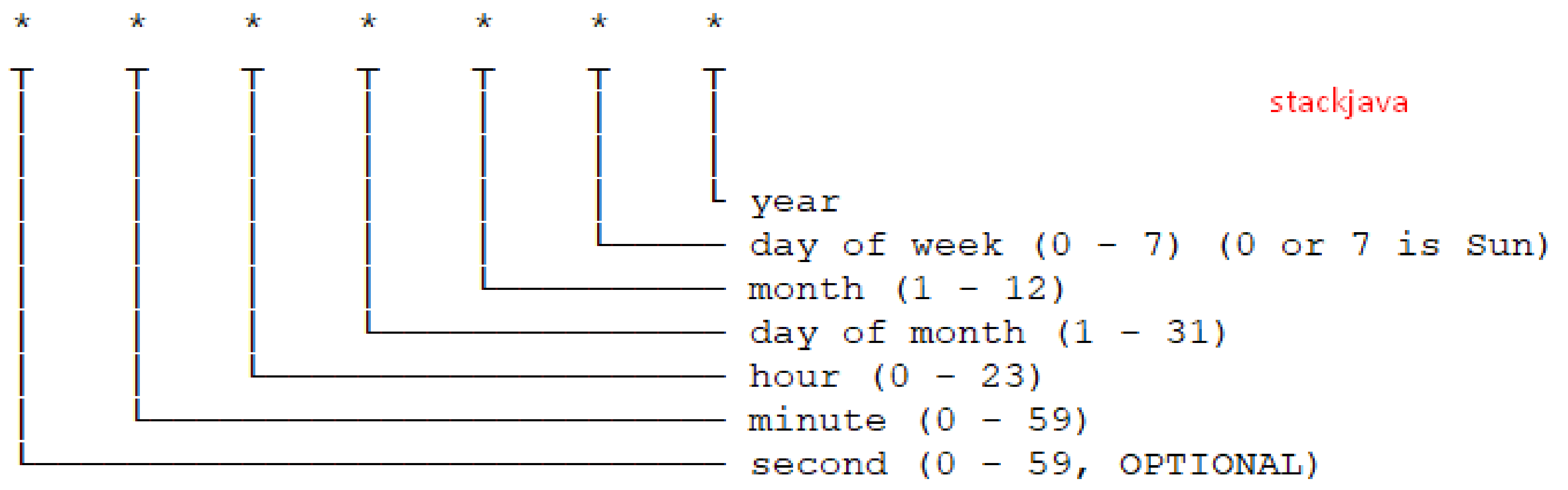
Cron expressions provide a way to define more complex schedules. They consist of six fields: second, minute, hour, day of month, month, and day of week.

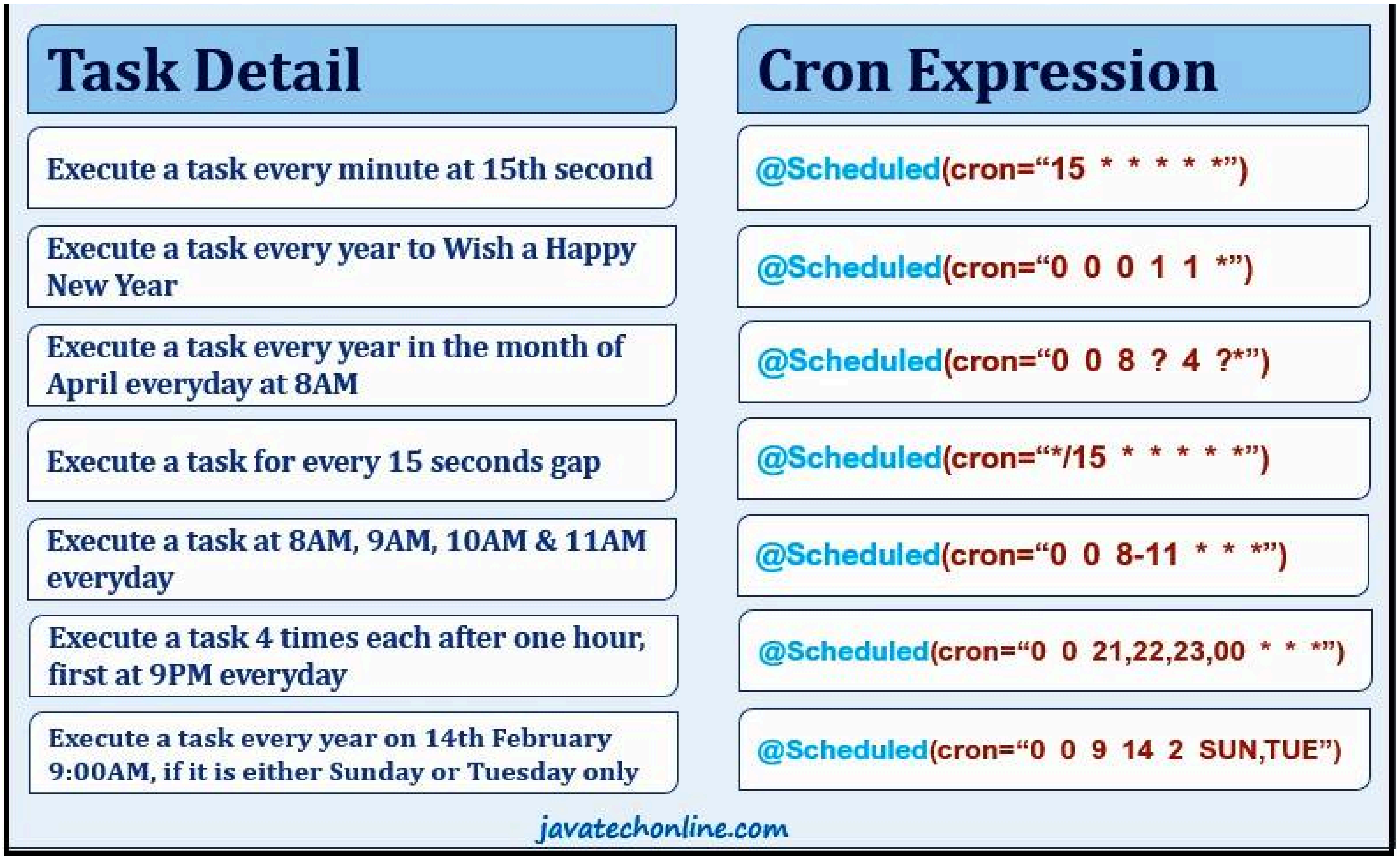


```
import lombok.extern.slf4j.Slf4j;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
@Slf4j
public class LogMyName {

    @Scheduled(cron = "0 * * * * *")
    public void sayHello(){

        log.info("Hello Rohan");
    }

}
```

This example runs the task after every minute.

Cron Expression





Task Detail	Cron Expression
Execute a task every minute at 15th second	@Scheduled(cron="15 * * * * *")
Execute a task every year to Wish a Happy New Year	@Scheduled(cron="0 0 0 1 1 *")
Execute a task every year in the month of April everyday at 8AM	@Scheduled(cron="0 0 8 ? 4 ?*")
Execute a task for every 15 seconds gap	@Scheduled(cron="*/15 * * * * *")
Execute a task at 8AM, 9AM, 10AM & 11AM everyday	@Scheduled(cron="0 0 8-11 * * *")
Execute a task 4 times each after one hour, first at 9PM everyday	@Scheduled(cron="0 0 21,22,23,00 * * *")
Execute a task every year on 14th February 9:00AM, if it is either Sunday or Tuesday only	@Scheduled(cron="0 0 9 14 2 SUN,TUE")

javatechonline.com

Advanced Scheduling Features

a. Parameterized Cron Expressions

You can **externalize cron expressions** to a configuration file:

```
scheduling.cron.expression=0 0 9 * * ?  # Every day at 9 AM
```

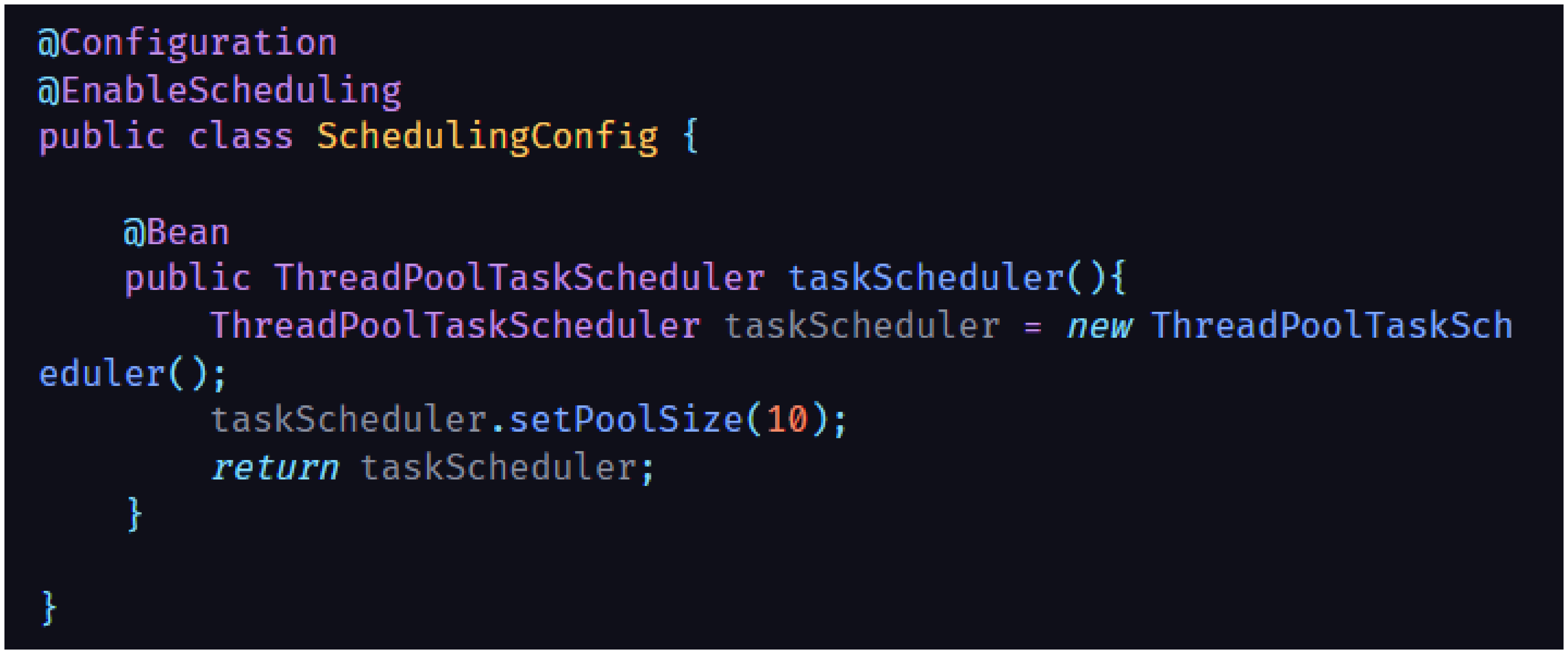
```
@Scheduled(cron = "${scheduling.cron.expression}")
public void performDynamicCronTask() {

    @Value("${scheduling.cron.expression}")
    private String cronExpression;


    @Scheduled(cron = "${scheduling.cron.expression}")
    public void generateDailyReport() {
        System.out.println("Generating daily report at scheduled tim
e: " + cronExpression);
    }
}
```

Advanced Scheduling Features

b. Asynchronous Scheduling

By default, tasks run in a **single-threaded executor**, meaning each scheduled task waits for the previous one to complete.

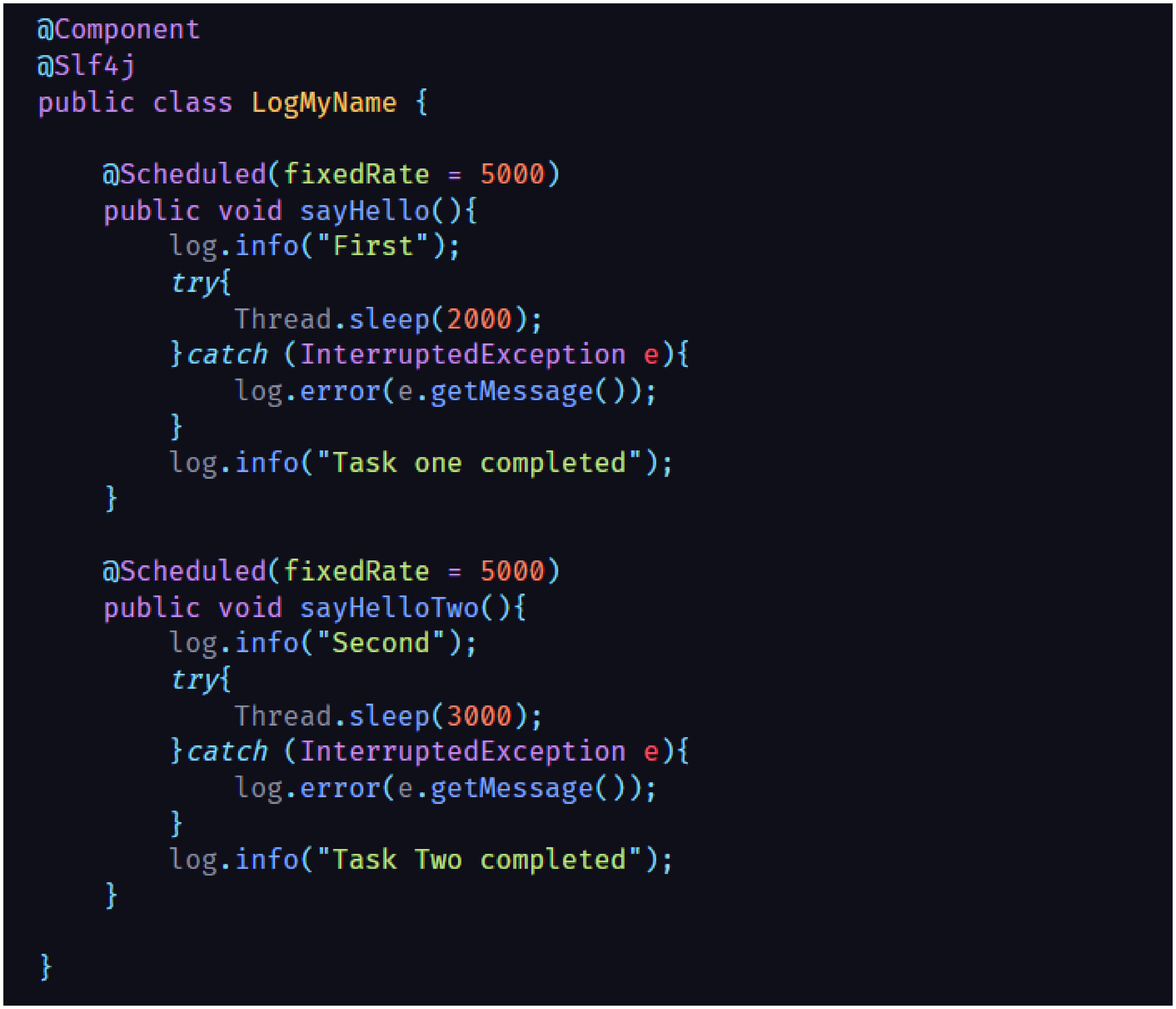
To allow **concurrent execution**, you can configure a **TaskScheduler bean** with a custom **thread pool**.



```java
@Configuration
@EnableScheduling
public class SchedulingConfig {

    @Bean
    public ThreadPoolTaskScheduler taskScheduler(){
        ThreadPoolTaskScheduler taskScheduler = new ThreadPoolTaskSch
eduler();
        taskScheduler.setPoolSize(10);
        return taskScheduler;
    }

}
```

Advanced Scheduling Features

b. Asynchronous Scheduling

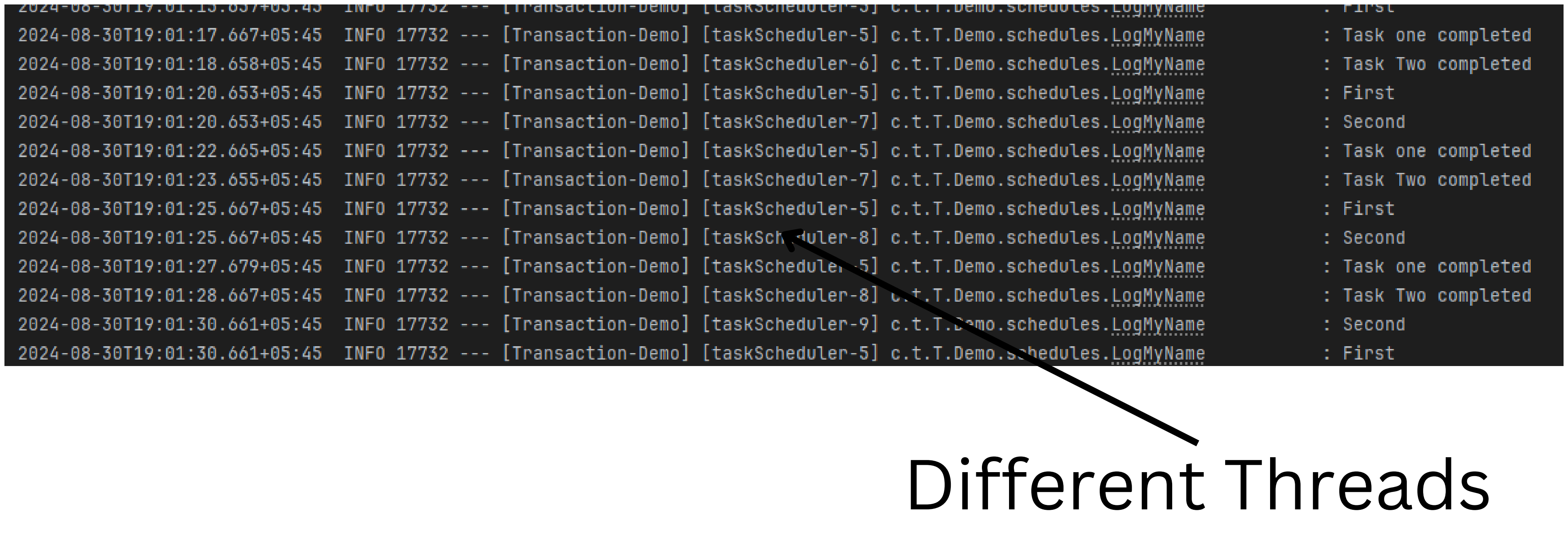


```
@Component
@Slf4j
public class LogMyName {

    @Scheduled(fixedRate = 5000)
    public void sayHello(){
        log.info("First");
        try{
            Thread.sleep(2000);
        }catch (InterruptedException e){
            log.error(e.getMessage());
        }
        log.info("Task one completed");
    }

    @Scheduled(fixedRate = 5000)
    public void sayHelloTwo(){
        log.info("Second");
        try{
            Thread.sleep(3000);
        }catch (InterruptedException e){
            log.error(e.getMessage());
        }
        log.info("Task Two completed");
    }

}
```

Advanced Scheduling Features

In this example:

- **sayHello()** and **sayHelloTwo()** are scheduled to run every **5 seconds**.
- Each task will run in its **own thread**, thanks to the **custom thread pool configured** earlier.
- The **Thread.sleep() simulates long-running** tasks, showing that they can run concurrently.



```
2024-08-30T19:01:15.657+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-5] c.t.T.Demo.schedules.LogMyName           : First
2024-08-30T19:01:17.667+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-5] c.t.T.Demo.schedules.LogMyName           : Task one completed
2024-08-30T19:01:18.658+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-6] c.t.T.Demo.schedules.LogMyName           : Task Two completed
2024-08-30T19:01:20.653+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-5] c.t.T.Demo.schedules.LogMyName           : First
2024-08-30T19:01:20.653+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-7] c.t.T.Demo.schedules.LogMyName           : Second
2024-08-30T19:01:22.665+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-5] c.t.T.Demo.schedules.LogMyName           : Task one completed
2024-08-30T19:01:23.655+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-7] c.t.T.Demo.schedules.LogMyName           : Task Two completed
2024-08-30T19:01:25.667+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-5] c.t.T.Demo.schedules.LogMyName           : First
2024-08-30T19:01:25.667+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-8] c.t.T.Demo.schedules.LogMyName           : Second
2024-08-30T19:01:27.679+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-5] c.t.T.Demo.schedules.LogMyName           : Task one completed
2024-08-30T19:01:28.667+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-8] c.t.T.Demo.schedules.LogMyName           : Task Two completed
2024-08-30T19:01:30.661+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-9] c.t.T.Demo.schedules.LogMyName           : Second
2024-08-30T19:01:30.661+05:45  INFO 17732 --- [Transaction-Demo] [taskScheduler-5] c.t.T.Demo.schedules.LogMyName           : First
```

Advanced Scheduling Features

c. Conditional Scheduling

You can conditionally enable or disable scheduled tasks based on application properties:

```
scheduling.enabled=true  # Set to false to disable scheduling
```

```java
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;

@Service
@ConditionalOnProperty(name = "scheduling.enabled", havingValue = "true", matchIfMissing = true)
public class EmailNotificationService {

    @Scheduled(cron = "0 0 12 * * ?")  // Every day at noon
    public void sendDailyNotifications() {
        System.out.println("Sending daily email notifications...");
        // Logic to send notifications
    }
}
```

Best Practices

- **Avoid Long-Running Tasks:** For long-running tasks, consider using a separate job scheduler like Quartz or a message queue.
- **Monitor Scheduled Tasks:** Implement monitoring to track scheduled task executions and failures.
- **Externalize Cron Expressions:** Externalizing cron expressions to a configuration file or database allows for dynamic changes without redeploying the application.
- **Use Dedicated Thread Pools:** For concurrent task execution, configure a dedicated thread pool to avoid blocking other tasks.

Conclusion

Spring Boot scheduling is a powerful tool for **automating routine tasks**. By leveraging the **@Scheduled** annotation, you can easily implement various scheduling strategies to meet your application's needs.

Thank You

Rohan Thapa

thaparohan2019@gmail.com