Spring Scheduling Annotations

**1. Overview**

When single-threaded execution isn’t enough, we can use annotations from the *org.springframework.scheduling.annotation* package.

In this quick tutorial, we’re going to explore the Spring Scheduling Annotations.

**2. *@EnableAsync***

With this annotation, we can enable asynchronous functionality in Spring.

We must use it with *@Configuration*:

|  |  |
| --- | --- |
| 1  2  3 | @Configuration  @EnableAsync  class VehicleFactoryConfig {} |

Now, that we enabled asynchronous calls, we can use *@Async* to define the methods supporting it.

**3. *@EnableScheduling***

With this annotation, we can enable scheduling in the application.

We also have to use it in conjunction with *@Configuration*:

|  |  |
| --- | --- |
| 1  2  3 | @Configuration  @EnableScheduling  class VehicleFactoryConfig {} |

As a result, we can now run methods periodically with *@Scheduled*.

**4. *@Async***

We can define methods we want to **execute on a different thread**, hence run them asynchronously.

To achieve this, we can annotate the method with *@Async*:

|  |  |
| --- | --- |
| 1  2  3  4 | @Async  void repairCar() {      // ...  } |

If we apply this annotation to a class, then all methods will be called asynchronously.

Note, that we need to enable the asynchronous calls for this annotation to work, with *@EnableAsync*or XML configuration.

More information about *@Async* can be found in [this article](https://www.baeldung.com/spring-async).

**5. *@Scheduled***

If we need a method to **execute periodically**, we can use this annotation:

|  |  |
| --- | --- |
| 1  2  3  4 | @Scheduled(fixedRate = 10000)  void checkVehicle() {      // ...  } |

We can use it to execute a method at **fixed intervals**, or we can fine-tune it with **cron-like expressions**.

*@Scheduled* leverages the Java 8 repeating annotations feature, which means we can mark a method with it multiple times:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Scheduled(fixedRate = 10000)  @Scheduled(cron = "0 \* \* \* \* MON-FRI")  void checkVehicle() {      // ...  } |

Note, that the method annotated with *@Scheduled* should have a *void* return type.

Moreover, we have to enable scheduling for this annotation to work for example with *@EnableScheduling* or XML configuration.

For more information about scheduling read [this article](https://www.baeldung.com/spring-scheduled-tasks).

**6. *@Schedules***

We can use this annotation to specify multiple *@Scheduled* rules:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @Schedules({    @Scheduled(fixedRate = 10000),    @Scheduled(cron = "0 \* \* \* \* MON-FRI")  })  void checkVehicle() {      // ...  } |

Note, that since Java 8 we can achieve the same with the repeating annotations feature as described above.

**7. Conclusion**

In this article, we saw an overview of the most common Spring scheduling annotations.