

# SPRING Scheduler



**UP ASI**  
**Bureau E204**

# Plan du Cours

- **Introduction**
- **@Scheduled Annotation**
- **Fixed delay vs Fixed rate**
- **Cron expression**

# Introduction

- La planification(scheduling) consiste à exécuter les tâches pendant une période de temps spécifique.
- Spring Boot Scheduling est une fonctionnalité pratique qui nous permet de planifier des tâches dans nos applications Spring Boot.
- Par exemple, si vous voulez que votre application exécute une tâche après un intervalle fixe ou en fonction d'un calendrier.
- **Le scheduler fait partie du module Core du framework Spring (Pas de dépendances à ajouter dans le pom.xml).**

# @Scheduled Annotation

- Spring Boot utilise l'annotation **@Scheduled** pour la planification des tâches.
- Il faut respecter certaines règles lors de l'utilisation de cette annotation :
  1. Les méthodes doivent être sans paramètre.
  2. Le type de retour de la méthode doit être void.
- Pour activer la planification (scheduling), il faut ajouter l'annotation **@EnableScheduling** à la classe main.

## **@EnableScheduling**

@SpringBootApplication


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

# @Scheduled Annotation

- Spring Boot permet de créer facilement une tâche de planification(scheduling task).
- Il suffit d'annoter la méthode avec l'annotation **@Scheduled**.
- L'annotation **@Scheduled** définit la planification (par exemple, quand la méthode sera exécutée, etc.).
- Nous pouvons passer certains paramètres à l'annotation pour personnaliser le comportement.

# Fixed Rate

- Pour planifier un déclenchement de méthode à une date fixe, nous pouvons utiliser le paramètre **fixedRate** dans l'annotation **@Scheduled**.
- Fixed Rate : permet à Spring d'exécuter la tâche à des intervalles périodiques, même si la dernière exécution est en cours.




exécuter la méthode  
toutes les 60 secondes

```
@Scheduled(fixedRate = 60000)
public void fixedRateMethod() {
    system.out.println("Method with fixed Rate");
}
```

# Fixed Delay

- Pour fixer un délai fixe entre la dernière exécution et le début de l'exécution suivante, nous pouvons utiliser le paramètre **fixedDelay**.
- Ce paramètre compte le délai **après l'exécution de la dernière invocation**.

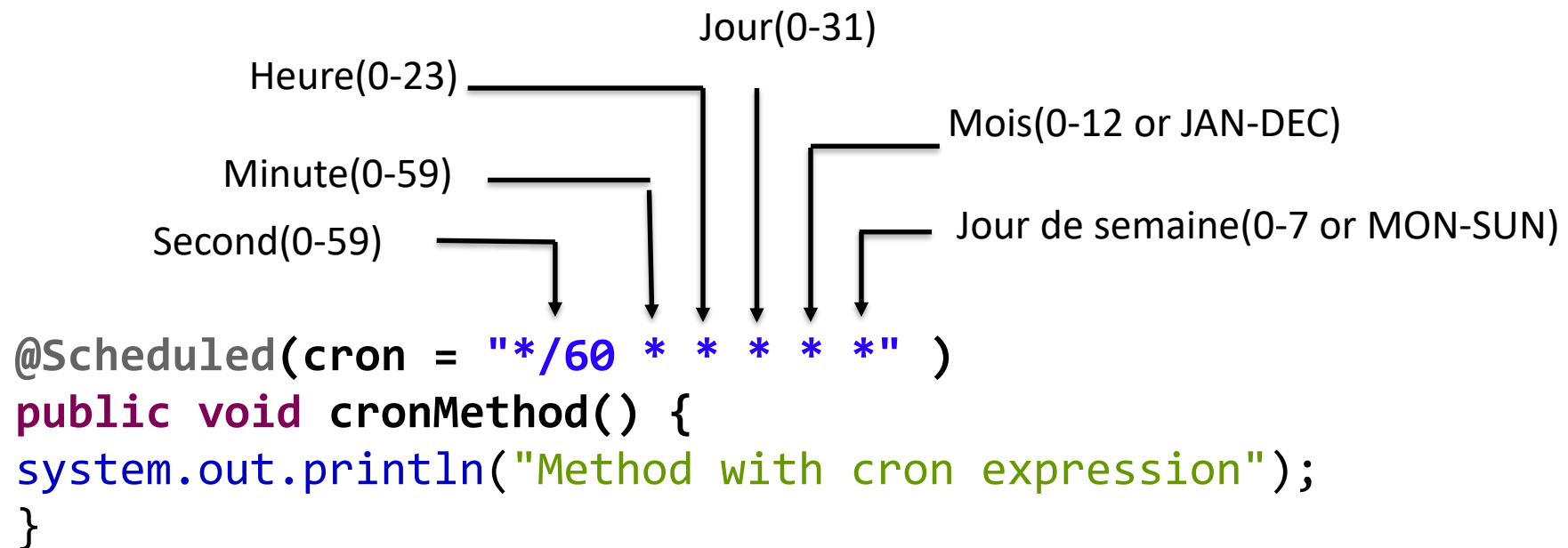


Les tâches sont  
déclenchées avec un  
**retard de 60 secondes.**

```
@Scheduled(fixedDelay = 60000)
public void fixedDelayMethod() {
    system.out.println("Method with fixed delay");
}
```

# Cron expression

- L'expression **Cron** est une façon **flexible** et **puissante** pour planifier les tâches.





# Exercise

1. `@Scheduled(cron = "15 * * * * *")`
2. `@Scheduled(cron = "*/15 * * * * *")`
3. `@Scheduled(cron = "0/15 * * * * *")`
4. `@Scheduled(cron = "0 0/30 11 * * *")`
5. `@Scheduled(cron = "0 0 8 ? 4 ?")` or `(cron = "0 0 8 * 4 *")`
6. `@Scheduled(cron = « 0 0 9 14 2 SUN,TUE »)`

# Conclusion

field	Description
FixedRate	Nombre de millisecondes après l'heure de début de la dernière exécution de la méthode. => @retourne la période en millisecondes.
FixedDelay	Nombre de millisecondes après la dernière exécution de la méthode. => @retourne le délai en millisecondes.
cron	Écrire le cron et définir le calendrier. Vous pouvez également spécifier le fuseau horaire. => @retourne une expression qui peut être analysée dans un plan de cron

# Travail à faire

Sur le même projet (magasin/stock) :

- Implémenter une méthode qui affiche le statut d'un stock pour chaque intervalle de 60 secondes.(en cas de rupture de stock)
- Calculer les revenus du magasin au début de chaque mois et de chaque année (1er janvier).

# SPRING Scheduler

Si vous avez des questions, n'hésitez pas à nous contacter :

**Département Informatique**  
**UP Architectures des Systèmes d'Information**  
Bureau E204