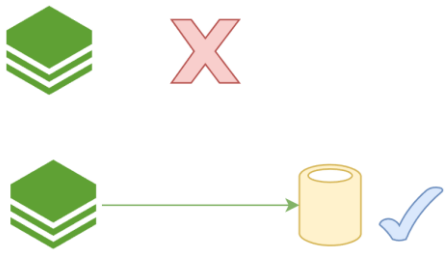
Como utilizar o Spring Batch?

Como um ***projeto Spring***, a maneira mais fácil de implementa-lo é adicionando ao gerenciador de dependências do projeto ***(maven/gradle),*** deixando isso a cargo do Spring boot fica bem mais fácil.

O primeiro ponto do **Spring Batch**, é que ele PRECISA de um **banco de dados**, isso porque ele grava metadados que são importantes pros processos que ele vai executar *(óbvio que H2 não é o indicado)*:



```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-batch</artifactId>
    </dependency>

    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
```

Outra configuração importante é a **habilitação do Spring Batch** que pode ser feita em uma **classe de configuração que contém o job** ou diretamente na **classe principal que contém o main** (se fizer direto na main ele habilita para todo o projeto):

```
@EnableBatchProcessing
@SpringBootApplication
public class LoteApplication {

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

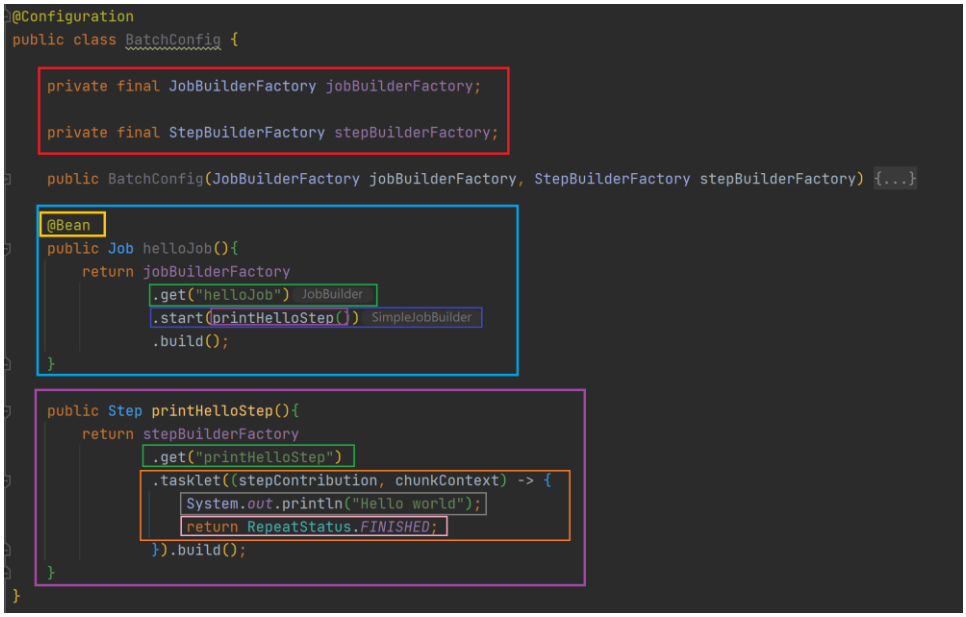
```
@EnableBatchProcessing
@Configuration
public class BatchConfig {
```

Escrevendo um Batch!

Sabemos que um **job** é uma rotina composta de **steps(pasos/etapas)**, o **Spring Batch** fornece **factorys e builders** para que um job e seus steps sejam construídos de maneira fácil e declarativa. Nesse exemplo eu posso usar uma classe pra representar isso.

Então eu injeto as **factorys** e através delas retorno o **Job** e o **Step** para serem executados, note que o **Job precisou ser anotado como @Bean** enquanto o **Step não**, isso porque o ***Step faz parte do Job***, então **declarando o Job como um Bean do Spring ele já consegue se virar com o gerenciamento do Step.**

Os ***.get() dão nome ao objeto (nesse caso tanto o job como o step)***, o **.start do job recebe o Step a ser executado**. O Step é do tipo ***TaskLet(quando uma tarefa é simples, esse é o tipo mais indicado pra sua execução)*** que além de conter o ***comportamento do Step***, também ***contém o status que deve retornar ao ser executado (nesse caso FINISHED):***


@Configuration
public class BatchConfig {
private final JobBuilderFactory jobBuilderFactory;
private final StepBuilderFactory stepBuilderFactory;
public BatchConfig(JobBuilderFactory jobBuilderFactory, StepBuilderFactory stepBuilderFactory) {...}
@Bean
public Job helloJob(){
return jobBuilderFactory
.get("helloJob")
JobBuilder
.start(printHelloStep())
SimpleJobBuilder
.build();
}
public Step printHelloStep(){
return stepBuilderFactory
.get("printHelloStep")
.tasklet((stepContribution, chunkContext) -> {
System.out.println("Hello World");
return RepeatStatus.FINISHED;
}).build();
}
}