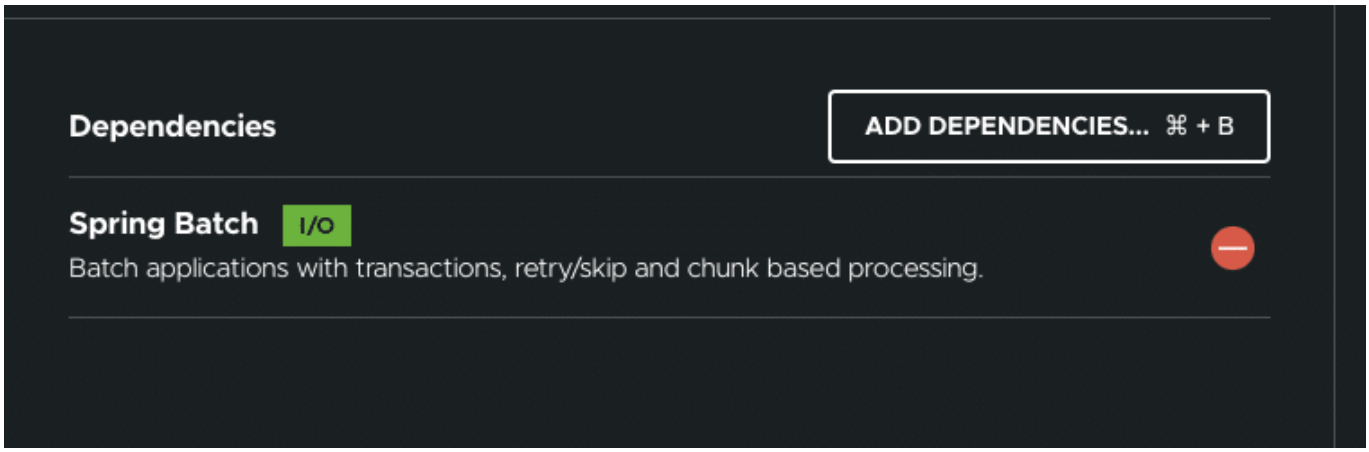
Acabamos de ver como usar Java equals y hashcode

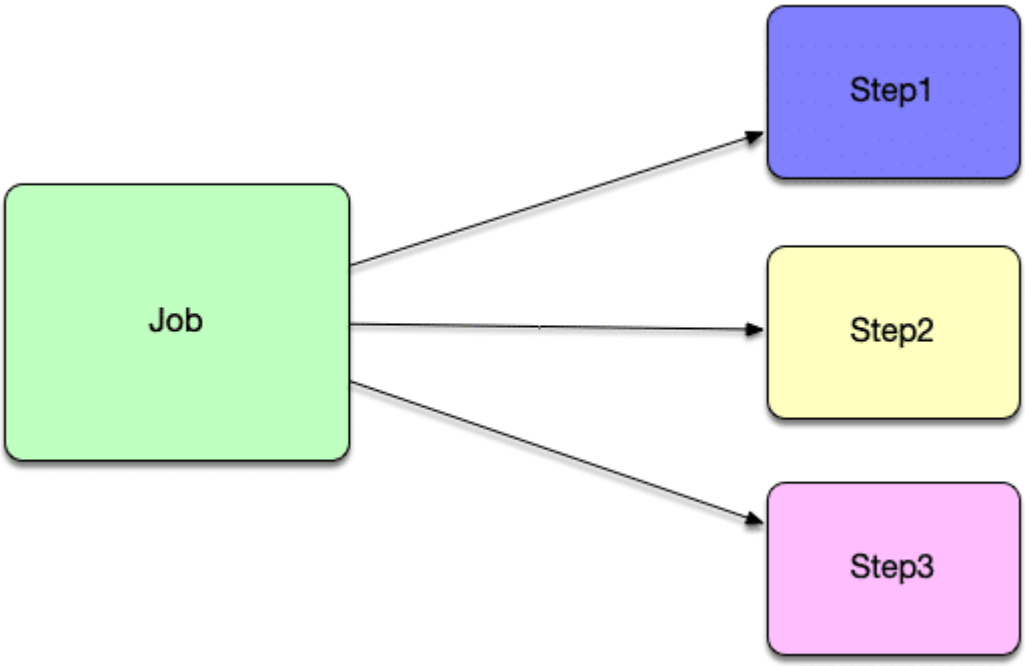
Spring Batch Hello World . Vamos a construir un ejemplo de introducción a Spring Batch . Batch es uno de los frameworks de Spring más especializados y se encarga de automatizar los procesos batch . Procesos que se realizan normalmente de forma desatendida en horarios nocturnos y que implican varias horas de procesamiento.

# Spring Boot y Spring Batch

Hoy por hoy cada día es más común desplegar los procesos batch apoyándonos en Spring Boot . Para ello nos es suficiente conectarnos a Spring initializr y generar un proyecto con la dependencia de Batch

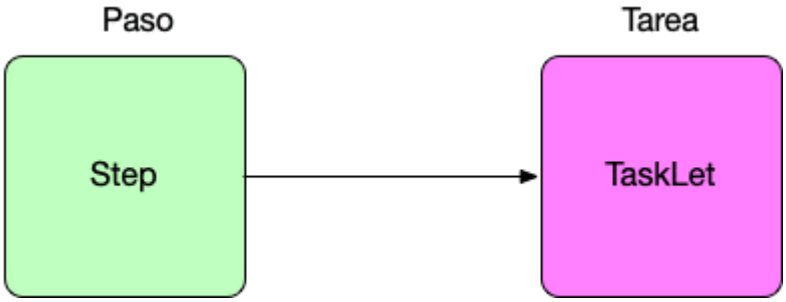



Batch permite una configuración muy amplia y aglutina muchos conceptos . Nosotros en este POST vamos a abordar los más elementales . Cuando nosotros desarrollamos un proceso Batch lo que realmente tenemos que realizar es un trabajo (Job) que dura un tiempo determinado . Ese trabajo normalmente esta dividido en pasos (steps) y cada paso realizará una tarea de terminada.




# Spring Batch Hello World y Steps

Cada uno de los steps debe ejecutar una tarea a estas tareas en Spring Batch se las denomina TaskLets.




Es momento de construir el código de cada uno de los TaskLet de tal forma que luego los podamos integrar en los Steps.

```
package com.arquitecturajava.batch1.steps;

import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.scope.context.ChunkContext;
```

```
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;

public class Tarea1 implements Tasklet {

	@Override
	public RepeatStatus execute(StepContribution contribution,
ChunkContext chunkContext) throws Exception {
		System.out.println("tarea1 ejecutado");
		return  RepeatStatus.FINISHED;
	}

}
```

```
package com.arquitecturajava.batch1.steps;

import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;

public class Tarea2 implements Tasklet {

	@Override
	public RepeatStatus execute(StepContribution contribution,
ChunkContext chunkContext) throws Exception {
		System.out.println("tarea2 ejecutado");
		return  RepeatStatus.FINISHED;
	}

}
```

```
package com.arquitecturajava.batch1.steps;

import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;

public class Tarea3 implements Tasklet {

        @Override
        public RepeatStatus execute(StepContribution contribution,
ChunkContext chunkContext) throws Exception {
                System.out.println("tarea3 ejecutado");
                return  RepeatStatus.FINISHED;
        }

}
```

Creadas las tareas , es momento de integrarlas en nuestro ejemplo de Spring Batch HelloWorld como pasos esto se puede realizar en un fichero que configura el Job y los pasos que tiene vamos a verlo:

```
package com.arquitecturajava.batch1;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import
org.springframework.batch.core.configuration.annotation.EnableBatchPro
cessing;
import
org.springframework.batch.core.configuration.annotation.JobBuilderFact
```

```
ory;
import
org.springframework.batch.core.configuration.annotation.StepBuilderFac
tory;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.arquitecturajava.batch1.steps.Tarea1;
import com.arquitecturajava.batch1.steps.Tarea2;
import com.arquitecturajava.batch1.steps.Tarea3;

@Configuration
@EnableBatchProcessing
public class JobConfiguration {

        @Autowired
        private JobBuilderFactory jobBuilderFactory;
        @Autowired
        private StepBuilderFactory stepBuilderFactory;
        @Bean
        public Step step1() {
                return stepBuilderFactory.get("paso1").tasklet(new
Tarea1()).build();
        }
        @Bean
        public Step step2() {
                return stepBuilderFactory.get("paso2").tasklet(new
```

```
Tarea2()).build();
        }
        @Bean
        public Step step3() {
                 return stepBuilderFactory.get("paso3").tasklet(new
Tarea3()).build();
        }
        @Bean Job holaMundoJob() {
                return jobBuilderFactory.get("holamundoJob")
                                .start(step1())
                                .next(step2())
                                .next(step3())
                                .build();
        }
}
```

Una vez que tenemos este fichero que define los pasos y activa las capacidades de Batch con la anotación @EnableBatchProcessing el siguiente paso es integrarlo dentro de Boot algo que es trivial usando la anotación @Import.

```
package com.arquitecturajava.batch1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Import;

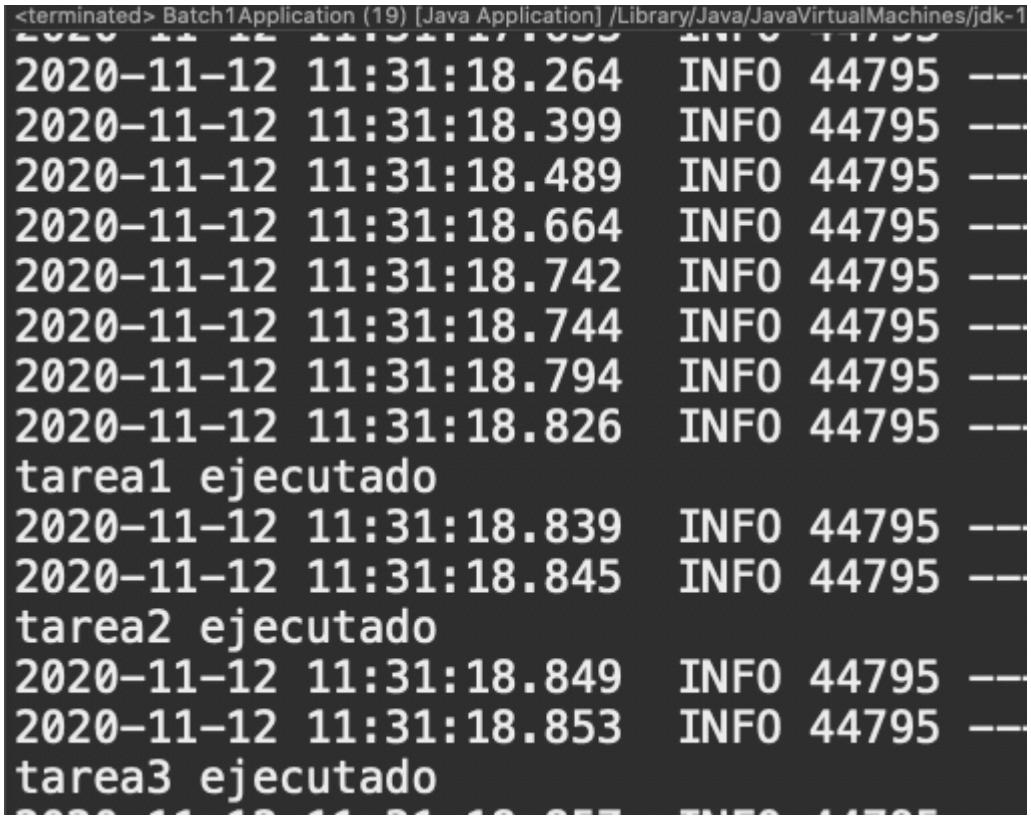
@SpringBootApplication
@Import(JobConfiguration.class)
public class Batch1Application {

        public static void main(String[] args) {
                SpringApplication.run(Batch1Application.class, args);
```

```
        }

}
```

Una vez configurado todo es momento de ejecutar la aplicación de Spring Boot.




Acabamos de ejecutar nuestro primer proceso batch usando Spring Batch con Spring Boot

## Otros artículos relacionados

- Spring Reactor y manejo de transformaciones
- ¿Qué es Spring WebFlux?
- Spring AOP annotation y su uso