**Spring Batch Listeners:**

* Spring Batch listeners are a way of intercepting the execution of a Job or a Step to perform some meaningful operations or logging the progress.
* We have the following types of event listeners which intercept the batch processing at specific events.
* JobExecutionListener (before and after job)
* StepExecutionListener (before and after step)
* ChunkListener
* ItemReadListener
* ItemProcessListener
* ItemWriteListener
* SkipListener

1**. JobExecutionListener**: [JobExecutionListener](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/JobExecutionListener.html) provides callbacks at before start and after completion of a [Job](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/Job.html).

**1.1. ImplimentingJobExecutionListener**

It should be noted that the afterJob() method is called regardless of the success or failure of the Job.

import org.springframework.batch.core.JobExecution;  
import org.springframework.batch.core.JobExecutionListener;  
  
public class JobResultListener implements JobExecutionListener {  
  
 public void beforeJob(JobExecution jobExecution) {  
 System.*out*.println("Called beforeJob().");  
 }

public void afterJob(JobExecution jobExecution) {  
 if (jobExecution.getStatus() == BatchStatus.COMPLETED ) {  
 //job success  
 }  
 else if (jobExecution.getStatus() == BatchStatus.FAILED) {  
 //job failure  
 }  
 }  
}

**1.2. Configuring JobExecutionListener**

@Bean  
public Job demoJob(){  
 return jobs.get("demoJob")  
 .incrementer(new RunIdIncrementer())  
 .listener(new JobResultListener())  
 .start(stepOne())  
 .next(stepTwo())  
 .build();  
}

2. **StepExecutionListener**: [StepExecutionListener](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/StepExecutionListener.html) allows for notification before a Step is started and after it ends, whether it ended normally or failed.

**2.1. Implementing StepExecutionListener**

import org.springframework.batch.core.ExitStatus;  
import org.springframework.batch.core.StepExecution;  
import org.springframework.batch.core.StepExecutionListener;  
  
public class StepResultListener implements StepExecutionListener {  
  
 @Override  
 public void beforeStep(StepExecution stepExecution) {  
 System.*out*.println("Called beforeStep().");  
 }  
  
 @Override  
 public ExitStatus afterStep(StepExecution stepExecution) {  
 System.*out*.println("Called afterStep().");  
 return stepExecution.getStatus();  
 }  
}

ExitStatus is the return type of afterStep in order to allow listeners the chance to modify the exit code that is returned upon completion of a Step.

**2.2. Configuring StepExecutionListener**

@Bean  
public Step stepOne(){  
 return steps.get("stepOne")  
 .tasklet(new MyTaskOne())  
 .listener(new StepResultListener())  
 .build();  
}  
  
@Bean  
public Step stepTwo(){  
 return steps.get("stepTwo")  
 .tasklet(new MyTaskTwo())  
 .listener(new StepResultListener())  
 .build();  
}

**3. ChunkListener:**

A chunk is defined as the items processed within the scope of a transaction. Committing a transaction, at each commit interval, commits a ‘chunk’.

[ChunkListener](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/ChunkListener.html) can be used to perform logic before a chunk begins processing or after a chunk has been completed successfully.

**3.1. Implementing ChunkListener:**

public class CustomChunkListener implements ChunkListener {  
 @Override  
 public void afterChunk(ChunkContext context) {  
 System.out.println("Called afterChunk().");  
 }  
  
 @Override  
 public void beforeChunk(ChunkContext context) {  
 System.out.println("Called beforeChunk().");  
 }  
  
 @Override  
 public void afterChunkError(ChunkContext context) {  
 System.out.println("Called afterChunkError().");  
 }  
}

The beforeChunk method is called after the transaction is started but before read is called on the ItemReader.

The afterChunk is called after the chunk has been committed and there is no rollback.

**3.2. Configuring ChunkListener:**

@Bean  
public Step stepOne(){  
 return steps.get("stepOne")  
 .tasklet(new MyTaskOne())  
 .listener(new CustomChunkListener())  
 .build();  
}

**4. ItemReadListener:**[ItemReadListener](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/ItemReadListener.html" \t "_blank) provides methods invoked around the reading of an item. Read listeners are beneficial to log the skipped records so that skipped records can be dealt with later.

**4.1. Implementing ItemReadListener:**

import org.springframework.batch.core.ItemReadListener;  
  
public class StepItemReadListener implements ItemReadListener<String> {  
  
 @Override  
 public void beforeRead() {  
 System.out.println("ItemReadListener - beforeRead");  
 }  
  
 @Override  
 public void afterRead(String item) {  
 System.out.println("ItemReadListener - afterRead");  
 }  
  
 @Override  
 public void onReadError(Exception ex) {  
 System.out.println("ItemReadListener - onReadError");  
 }  
}

The beforeRead method is called before each call to read on the ItemReader.

The afterRead method is called after each successful call to read and is passed the item that was read.

If there was an error while reading, the onReadError method is called.

**4.2. Configuring ItemReadListener:**

@Bean  
public Step stepOne(){  
 return steps.get("stepOne")  
 .tasklet(new MyTaskOne())  
 .listener(new StepItemReadListener())  
 .build();  
}

**5. ItemProcessListener:** [ItemProcessListener](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/ItemProcessListener.html) provides methods invoked around the processing of an item. Implementations of this interface will be notified before and after an item is passed to the [ItemProcessor](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/item/ItemProcessor.html) and in the event of any exceptions thrown by the processor.

**5.1. Implementing ItemProcessListener:**

import org.springframework.batch.core.ItemProcessListener;  
public class StepItemProcessListener implements ItemProcessListener<String, Number> {  
  
 @Override  
 public void beforeProcess(String item) {  
 System.out.println("ItemProcessListener - beforeProcess");  
 }  
  
 @Override  
 public void afterProcess(String item, Number result) {  
 System.out.println("ItemProcessListener - afterProcess");  
 }  
  
 @Override  
 public void onProcessError(String item, Exception e) {  
 System.out.println("ItemProcessListener - onProcessError");  
 }  
}

**5.2.** **Configuring ItemProcessListener:**

@Bean  
public Step stepOne(){  
 return steps.get("stepOne")  
 .tasklet(new MyTaskOne())  
 .listener(new StepItemProcessListener())  
 .build();  
}

**6. ItemWriteListener:** [ItemWriterListener](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/ItemWriteListener.html) is used for events notified before, after, and in case of any exception thrown while writing a list of items.

**6.1. Implementing ItemWriteListener:**

import java.util.List;  
import org.springframework.batch.core.ItemWriteListener;  
public class StepItemWriteListener implements ItemWriteListener<Number> {  
  
 @Override  
 public void beforeWrite(List<? extends Number> items) {  
 System.out.println("ItemWriteListener - beforeWrite");  
 }  
  
 @Override  
 public void afterWrite(List<? extends Number> items) {  
 System.out.println("ItemWriteListener - afterWrite");  
 }  
  
 @Override  
 public void onWriteError(Exception exception, List<? extends Number> items) {  
 System.out.println("ItemWriteListener - onWriteError");  
 }  
}

**6.2. Configuring ItemWriteListener:**

@Bean  
public Step stepOne(){  
 return steps.get("stepOne")  
 .tasklet(new MyTaskOne())  
 .listener(new StepItemWriteListener())  
 .build();  
}

**7. SkipListener:**

[SkipListener](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/StepListener.html) listens to skipped items. Its methods will be called by [Step](https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/Step.html) implementations at the appropriate time in the step lifecycle.

**7.1. Implementing SkipListener:**

import org.springframework.batch.core.SkipListener;  
public class StepSkipListener implements SkipListener<String, Number> {  
  
 @Override  
 public void onSkipInRead(Throwable t) {  
 System.out.println("StepSkipListener - onSkipInRead");  
 }  
  
 @Override  
 public void onSkipInWrite(Number item, Throwable t) {  
 System.out.println("StepSkipListener - afterWrite");  
 }  
  
 @Override  
 public void onSkipInProcess(String item, Throwable t) {  
 System.out.println("StepSkipListener - onWriteError");  
 }  
}

**7.2. Configuring SkipListener:**

@Bean  
public Step stepOne(){  
 return steps.get("stepOne")  
 .tasklet(new MyTaskOne())  
 .listener(new StepSkipListener())  
 .build();  
}