**READERS:**

**ItemReader**

package org.springframework.batch.item;

public interface ItemReader<T> {

T read() throws Exception, UnexpectedInputException, ParseException,

NonTransientResourceException;

}

**PROCESSORS:**

From Minnella:

*Validate input*: In the original version of Spring Batch, validation occurred at the

ItemReader by subclassing the ValidatingItemReader class. The issue with this

approach is that none of the provided readers subclassed the

ValidatingItemReader class so if you wanted validation, you couldn’t use any of

the included readers. Moving the validation step to the ItemProcessor allows

validation to occur on an object before processing, regardless of the input method.

This makes much more sense from a division-of-concerns perspective.

*Reuse existing services*: Just like the ItemReaderAdapter you looked at in Chapter 7

to reuse services for your input, Spring Batch provides an ItemProcessorAdapter

for the same reason.

*Chain ItemProcessors*: There are situations where you will want to perform

multiple actions on a single item within the same transaction. Although you could

write your own custom ItemProcessor to do all of the logic in a single class, that

couples your logic to the framework, which is something you want to avoid.

Instead, Spring Batch allows you to create a list of ItemProcessors that will be

executed in order against each item.

1. **Validazione Customers**
   1. **ValidatingItemProcessor**
2. ItemReaderAdapter  
   (delegate a business service da config)
3. CompositeItemProcessor
   1. Leggi tabelle specifiche
   2. Sulla base dei dati letti fai un calcolo
   3. Risultato usato da prossimo Processor nella chain

**WRITERS:**