**DEMO CHUNK ORIENTED NOTE:**

Demo Bonifici:BonificoDto.  
Copia dati con Log processing da source a destination.

**DEMO:**

1. Copia formato
   * 1. Csv
     2. Xml
     3. Jdbc
        1. Cursore
        2. Paginato
2. Delega a servizi (Adapters)
   1. Reader
   2. Processor
   3. Writer
3. Processors:
   1. Chain Processors
   2. Filter -> return null (quando il record deve essere saltato)
   3. Eccezioni
      1. Reader
      2. Processors
      3. Writer
      4. Fault Tolerance -> Design
4. Writers
   1. FlatFiles
      1. LineAggregators
         1. PassThrough
         2. FormatterLineAggregator (String.format)
         3. DelimitedLineAggregator (prop list, separator)
      2. MultiResourceItemWriter
   2. StaxEventItemWriter
   3. JdbcBatchItemWriter
   4. Adapters:
      1. PropertyExtractingDelegatingItemWriter
5. Eccezioni:
   1. Skippable
      1. Obiettivo: Log o save Record skip
      2. Reader => non in Processor
      3. Processor => non in Writer
      4. Writer => non in Output
   2. Retriable (anche skippable)
      1. Status Riprendere Page e Record.
      2. Reader => try fino a limite
      3. Processor
         1. Problema Fault tolerance elaborazione parziale
            1. Immutable input e return processed copy
            2. Flag per elaborazione (batch DTO wrapper)
         2. Ri- Elaborazione page intera (Immutable)
         3. Ri- Elaborazione singolo record (Flag)
      4. Writer
         1. Uguale a Processor. Se va in RE riprova dal Processor.
   3. Rollback
      1. Rollback e sistemi esterni (reader, processor, writer)

**NOTE DEMO:**

Punti di manovra demo:  
1) COP-Demo.xml (commentare import file configurazione)  
2) Launcher (per csv e xml modificare path)

Init:  
1) Creare tabella su DB e insert dati (\setup\Payments.sql)  
2) Creare/Synch dirs come da path in Launcher

**TEST DEMO:**

1. Sbatch-COP-01-FlatFiles.xml:
   1. Test IO:
      1. Verifica sul path del Launcher , in, out e log del processor
   2. Test Basic Configs
      1. Reader dependencies
      2. Writer dependencies
   3. Test Eccezioni
      1. Vedi pattern
2. Sbatch-COP-02-Xml.xml
   1. Test IO:
      1. Verifica sul path del Launcher , in, out e log del processor
   2. Test Basic Configs
      1. Reader dependencies
      2. Writer dependencies
   3. Test Eccezioni
      1. Vedi pattern
3. Sbatch-COP-03-Jdbc-Cursor.xml
   1. Test IO:
      1. Verifica sul path del Launcher , in, out e log del processor
   2. Test Basic Configs
      1. Reader dependencies
      2. Writer dependencies
   3. Test Eccezioni
      1. Vedi pattern
4. Sbatch-COP-04-Jdbc-Paging.xml
   1. Test IO:
      1. Verifica sul path del Launcher , in, out e log del processor
   2. Test Basic Configs
      1. Reader dependencies
      2. Writer dependencies
   3. Test Eccezioni
      1. Skippable,
      2. Retriable, (verifica order by per ripetere stessa page)

1. Sbatch-COP-05-Adapters.xml
   1. Test IO:
      1. Verifica sul path del Launcher , in, out e log del processor
      2. In debug read, process, write
   2. Test Basic Configs
      1. Reader dependencies
      2. Writer dependencies
   3. Test Eccezioni
      1. Skippable,
      2. Retriable,
      3. Rollback
      4. Runtime
2. Sbatch-COP-06-ChainedProcessors.xml:
   1. Test IO:
      1. Verifica sul path del Launcher , in, out e log del processor
      2. In debug read, process, write
      3. Demo filters (null)
   2. Test Basic Configs
      1. Reader dependencies
      2. Writer dependencies
   3. Test Eccezioni
      1. Skippable,
      2. Retriable,
      3. Rollback
      4. Runtime

1. Sbatch-COP-07-StreamReader.xml:
   1. Versione senza eccezioni.   
      Commentare chiamata a   
      BonificoItemStream.read

ExceptionDemoUtils.evaluateExceptions(readCount, this.getClass());

**NOTE:**

1. Per ogni tipo di Reader**,** Writer evidenziare le classi custom (package) e partire dallo specifico file di config
2. Per gli adapter provare a modificare i tipi io dei metodi (es oggetto singolo)  
   Sbatch fa quello che puo’ per interpretare, salta quello che non capisce.
3. Chained processor discorso sul design:  
   Scenario (Venditori-> Ordini -> Items (provvigioni diverse)  
     
   Design processors:
   1. Dati
      1. Enrichment copia immutabile bean read (vedi fault tolerance)
      2. Oggetto Wrapper che viene popolato ad ogni processor
      3. Oggetti di input per prossimo Processor
   2. Problemi ricorrenti:
      1. Trasformazioni
      2. Validazioni
      3. Filter -> return null
      4. Aggregazione/Split

**DESIGN CHAIN PROCESSORS SCENARIO: (vedi scenari dettaglio)**

1. **Esempio Enrichment:**
   1. Fattura smartphone
2. **Esempio Wrapper:**
   1. Busta Paga
3. **Esempio Pipeline:**
   1. Validatori bonifici bancari
4. **Per ogni scenario**
   1. Filter (non processare un record)
   2. Fault tolerance
      1. Skip (e save)
      2. Retry (Page)
      3. Rollback

**Enrichment vs Wrapper:**

Sottile differenza basata su quantita di dati che e’ necessario scrivere.  
Si basano su problematiche in cui per ogni record ci sono N record figli da leggere.  
In lettura sono piu o meno uguali.   
La differenza (sottile) c’e’ nella scrittura.  
Se bisogna scrivere dati di entita diverse allora e’ Wrapper, se invece solo singoli risultati (di aggregazione ad es) siamo in un design di tipo Enrichment.

**CHAIN PROCESSORS FILTERS:**

1. Cosa accade se in pipeline validazione uno dei controlli non passa?  
   Scrivi in una tabella Bonifici bloccati.  
   Salta validatori successivi. -> Filter -> Null
2. Processor possono scrivere?
   1. Bisognerebbe evitare il piu possibile per necessita di fault tolerance sulla retry.  
      Casi eccezionali:  
      In casi di eccezione con logica transazioni requires\_new singolo record.  
      In tutti questi casi bisogna marcare come processato il record (su DB e in memoria sulla collection prodotta dal reader), per evitare rielaborazione in caso di repeat.

Fault Tolerance:

* 1. Gestione eccezioni deterministiche
     1. Page
        1. SkipPageLogic vs Fail Batch?
     2. Record:
        1. SkipRecord vs Fail
  2. Gestione eccezioni transitorie:
     1. Page (servizio NA)
     2. Record: Skip, o save
  3. Retry
     1. Reader
     2. Processor
     3. Writer

**FAULT TOLERANCE:**

Comune a tutte le demo.

- Retry ripete l’operazione (read, process, write) su tutta la Page di dati (commit-interval)  
- Skip salta (non ripete) l’operazione (read, process, write) sul record che e’ andato in eccezione.  
  
  
**Eccezione nel Reader:**

**Skippable/Retriable**

Sul reader solo la Skippable e’ supportata.  
Dal momento in cui e’ il singolo read<T> che fallisce, saltare il record equivale ad una retry.  
  
Per cui:  
- L’unico possibile intervento e’ rendere l’eccezione Skippable se transitoria.  
- Se un’eccezione lanciata nel reader non e’ skippable il Job fallisce.  
- Se non transitoria il Job non puo’ che fallire.   
In tal caso importante la diagnostica dei log.

**RollbackException:**

- Attenzione a reader transazionali (es JMS).   
Fallire nel consumare un messaggio e gestione ack su MOM.

**Eccezione nel Processor:**I dati non verranno riletti dal reader.

**Retriable/On Repeat**

Il Processor ripetera’ la sua esecuzione (uno alla volta) sugli N record della collection in memoria.

* Problema esecuzione parziale. (Soluz -> restituire copia oggetto Immutabile)  
  Es. Page di 50 records.  
  Il n. 49 va in eccezione retriable.  
  Gli altri 48 sono stati gia modificati.  
  Alla prossima esecuzione il processor agira’ sui 48 gia processati.  
    
  O in alternativa e’ importante considerare che modificare un oggetto della collection del reader, sopravvivera’ ad una retry, per cui si potrebbe avere uno stato inconsistente con alcuni record modificati ed altri invece ancora da processare.

**DEMO:**

1. Rendere probabile una Retriable Exception in ExceptionDemoUtils
2. Abilitare l’eccezione nel BonificoListProcessor
3. Tests
   1. Con BonificoListProcessor.updateAndReturnNew **(Safe)**
      1. Quando entra in retry la collection items e’ uguale all inizio.   
         E quindi non c’e’ il problema di ripulire le elaborazioni parziali.
   2. Con BonificoListProcessor.updateAndReturnSame **(unsafe)**
      1. Quando rientra nel metodo dopo la retry la collection e’ parzialmente elaborata (**vedi field description**).  
         Bisogna in tal caso gestire le elaborazioni parziali delle esecuzioni fallite.

**Skippable:**

Salta il record. Fino allo skip limit, poi fallisce il Job.  
Nel Writer non verra’ scritto.

**DEMO:**

1. Rendere probabile una Skippable Exception in ExceptionDemoUtils
2. Abilitare l’eccezione nel BonificoListProcessor
3. Verificare che il record andato in skip non viene scritto dal Writer.

**Writer:**

**Retriable/On Repeat:**Ripete dal processor su tutto il chunk in memoria.  
Superato il retry limit, applica la skip logic. (Se l’eccezione ’anche skipabble)

**Skippable:**Scompone la collection nei singoli elementi.Per ognuno di loro, in singola transazione requires\_newesegue process> write.  
Se si verifica un’eccezione il record corrente viene saltato e si passa al successivo.

**ITEM STREAM.**Interfaccia che definisce i metodi

1. open => inizializzazione risorse
2. update =>prima di ogni ciclo di read
3. close => cleanup risorse

Close (fine esecuzione reader) => read return null

Q:  
**Ci passa solo dopo commit?**  
  
**AbstractItemCountingItemStreamItemReader** ?  
da estendere migliora la logica utilizzando un contatore che permette di saltare i record gia processati/scritti e riprendere in caso di Retry/Skip direttamente dal successivo a quello andato in eccezione.

**Verifica estensioni esistenti:**

**AbstractItemCountingItemStreamItemReader**  
Contatori:

Ha un contatore ed un massimo.  
Ad ogni read completata aumenta il contatore.  
Se il contatore supera il max return null.

Open():

1. Se saveState
   1. Legge lo stato corrente del contatore e del max dal EC.
   2. Verifica chiavi su contesto EC:

Se non trova le chiavi (prima esecuzione) li inizializza entrambi a zero e le mette nel EC.

Se trova le chiavi vuol dire che siamo in una restart.  
Se l item count e’ >0 e <max allora fa uno skip di N elementi col metodo jumpToItem(itemCount); che di default esegue <itemCount> volte il read()

Read():

Aumenta il contatore.  
Esegue il doRead() delegato alle subclassi  
Associa al T restituito il contatore. (vedi ItemCountAware)

Update():

Se saveState aggiorna le chiavi nel EC. (contatore e max)

close():

Azzera il contatore ed esegue il doClose()

**StaxEventItemReader**

**doOpen():**

**Inizializza risorsa**

**doRead():**

**prova a trovare un nuovo fragment (primo livello sotto root) Es. catalogo/cd**

**se lo trova lo parsa col marshaller in un oggetto (es. cd)**

**doClose():**

**cleanup risorsa**

**ITEM STREAM READER:  
retry non supportato su readers.  
Skip in caso di fallimento del read(). (dovrebbe ottenere stesso risultato)**

1. Ciclo esecuzione
   1. Open => inizio step
   2. Update =>prima di ogni ciclo di read
   3. Read => N fino a commit-interval
   4. Close (fine esecuzione reader) => read return null
2. EccezioniGestione stato:
   1. Repeat: Necessario salvare stato di quante volte e’ andato in esecuzione.  
      Accesso allo ExecutionContext

**ITEM STREAM WRITER**

**Design**

1. Open (start o restart)
   1. Init risorse
   2. Gestione retry e stato:
      1. Leggere da EC ultimo record processato.
      2. Se c’e’ (e’ un restart) ripartire da li invece che   
         ripetere process-write su tutta collection
2. Update:
   1. Update stato. (ultima commit completata con successo)  
      Cio che e’ necessario per restore in caso di repeat
3. Close()
   1. Cleanup risorse. Azzeramento contatori.

**ESERCIZIO: REST CLIENT ITEM STREAM READER**

**Scenario:**Chiamata a servizio rest con paginazione.  
  
**Requisiti:**1) Ripetere chiamata in caso di servizio non disponibile (potrebbe essere transitorio)  
2) Commit ad ogni pagina ricevuta.  
2.1) Eccezione retriable (SNAException)  
Eccezioni skippable (mapping exception sul singolo record)  
2.2) Ripetizioni con 3 secondi di delay in caso di eccezione SNAException  
Retry 5 volte.  
2.2) Salvare quale record quale page per <SkippableException>

Nel caso in cui sia necessario creare un’implementazione di un ItemReader, la cosa piu ricorrente e’ che venga utilizzato un ItemStreamReader.  
Per questi motivi:

1. Mantenimento dello stato tra una esecuzione e la successiva
2. Gestione ripetizioni esecuzione. (fallisce al te

Scenario:

Requisiti:

**Soluzione:**

BonificoItemStream  
ComponenteHttp. (init, close)

Metodi:  
  
1) BonificoItemStream .open(): init del componente -> ComponenteHttp.init  
2) BonificoItemStream .close(): cleanup componente -> ComponenteHttp.close  
3) BonificoItemStream .read(): gestione singolo read. Lettura dal EC in merito a esecuzioni precedenti.

**Logica downloadPage:**  
1) load dati nella response.  
2) Restituisce Page sottoforma di List<BonificoBean>  
3) Se timeout restituisce SNAException  
4) Salva (per test log) pagina su path.