Java Batch

Vorstellungsrunde

- Name und die Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Aktuelle Problemsituation
- Individuelle Zielsetzung

https://integrata-cegos.deskmate.me/

integrata cegos

tn14.raum01@integrata-cegos.de 3338_tn14 administrator

deumeniwako

tn15.raum01@integrata-cegos.de 3338_tn15 administrator

niehoff

tn16.raum01@integrata-cegos.de 3338_tn16 administrator

opitz

tn17.raum01@integrata-cegos.de 3338_tn17 administrator

varga

Ausgangssituation

Programmiersprache Java

- Sequentielle Programmausführung
- Variablen werden im Hauptspeicher abgelegt und sind damit ausschließlich in einem laufenden Java-Prozess (Java Virtual Machine) bekannt
  - Persistente Datenablage muss gesondert programmiert werden
    - entityManager.persist(object)
- Ressourcen-Zugriffe sind blockierend
- Parallelisierung ist nicht Bestandteil der Sprach-Syntax, sondern wird über Bibliotheken realisiert
  - executorService.run(() -> {//sequenz})

Java und Batch-Programmierung

integrata cegos

- Das passt so überhaupt nicht zusammen!
- Anforderungen an ein Batch-Programm
  - Parallelisierung über Prozesse hinweg
  - Einzelne Sequenzen, Steps, in einem Batch-Programm können im Fehlerfall automatisch wiederholt werden
  - Batch-Variablen sind immer persistent zu sehen
- Trend: Batch-Programmierung führt eigene Programmiersprachen ein
  - z.B. in Workflow-Sprachen
    - BPM 2.0
    - Proprietäre Sprachen für Workflow-Engines

Batch-Programmierung und Java

integrata cegos

- Verlangt den Einsatz eines Frameworks
  - Historie: Spring Community startet das Projekt “Spring Batch”
  - Java Enterprise Enterprise Community hat später ein Projekt “Java Batch” entwickelt (?)
    - Spring Batch wurde wurde 1:1 übernommen
      - Manche Klassen / Interfaces wurden umbenannt
      - Paketstruktur wurde angepast
    - In der Praxis benutzen die meisten Projekte Spring Batch
- Spring Batch kann kein Komplettpaket sein (!)
  - Notwendige zusätzliche Infrastruktur
    - Datenbank hält Batch-Variablen sowie zusätzliche Status-Informationen
    - Orchestrierung der Batch-Prozesse “von außen”

Batch-Programmierung und Java

integrata cegos

- Das Programmmiermodell von Spring Batch ist
  - “nicht schön”
  - “ungewöhnlich”
  - “sperrig”
- Batch-Runtime ist relativ komplex
  - Java Batch ist Bestandteil der JEE
    - Runtime ist eine Applikationsserver, z.B. JBoss / Wildfly, WebSphere
  - Spring Batch kann als Standalone Spring Boot-Applikation betrieben

Spring Batch: First Contact

Abstraktion eines Batch-Jobs

- Ein Batch-Job wird abgekürzt als “Job” bezeichnet
- Jeder Job besteht aus einzelnen “Steps”
- Analogie zur sequenziellen Programmierung
  - Eine Funktion
  - besteht aus mehreren Anweisungen
  -

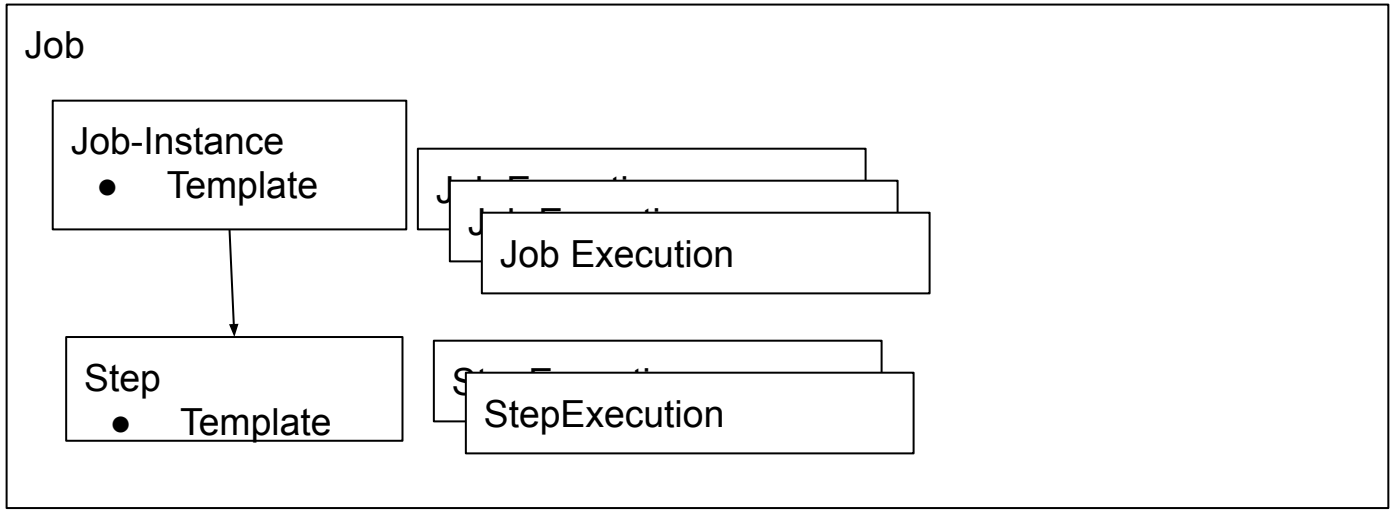
HelloWorldJobConfiguration revisited

integrata cegos

- @Configuration, @Autowired und @Bean
  - Annotations von Spring Boot
  - Beim Start der Application werden diese analysiert und ausgeführt
  - Haben keinen direkten Bezug zu Spring Batch
- @EnableBatchProcessing
  - Diese Annotation zwingt Spring Boot dazu, die nötige Infrastruktur für eine Batch-Ausführung bereitzustellen
    - JobBuilderFactory und StepBuilderFactory stehen erst nun zur Verfügung
    - Datenbank vom Typ der Embedded H2

Job, Step und Tasklet

Tasklet
- eine Hülle um beliebige Java-Sequenzen

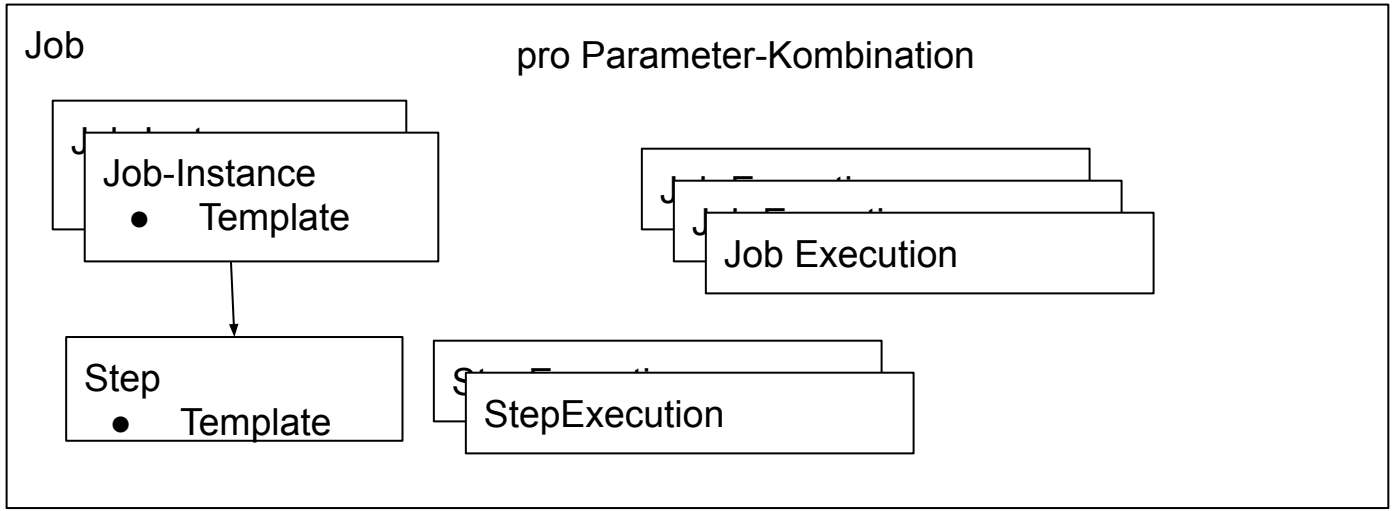


ToDo

- Stellen auch Sie die Anwendung auf eine externe Datenbank um
  - application.properties und startExternalH2.txt finden Sie auf GitHub
- Starten Sie die Anwendung gegebenenfalls mehrfach und betrachten Sie die Ausgaben in den Batch-Tabellen
- Ändern Sie den vorhandenen Job ab, z.B. neuer Name und neuer Step-Name

Job, Step und Tasklet

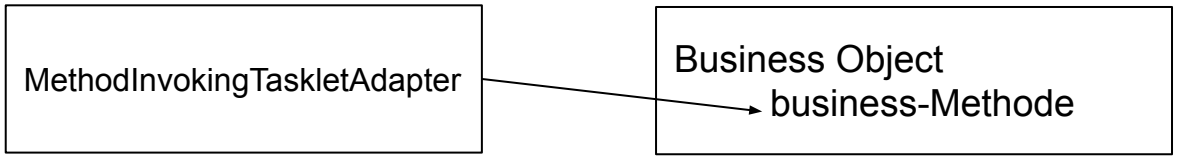
Tasklet
- eine Hülle um beliebige Java-Sequenzen



Job Programmierung

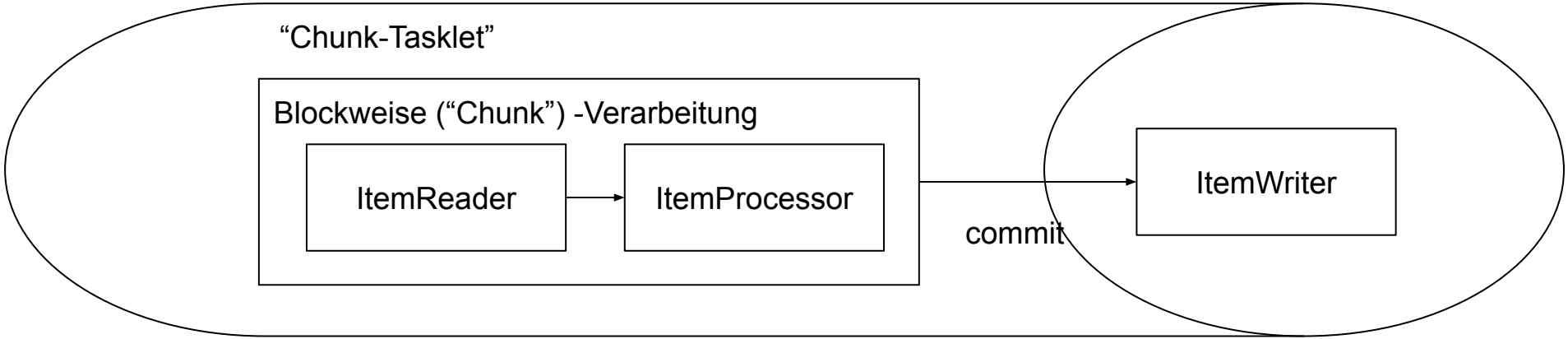
Tasklet

- Ein kleiner Wrapper um eine beliebige Java-Logik
- Einige wenige fertige Tasklets sind bereits vorhanden
  - CallableTaskletAdapter
    - Bisher wurde für die Ausführung der Tasklet-Logik der Job-Thread benutzt
  - MethodInvokingTaskletAdapter



  - SystemCommandTasklet
    - Ruft ein Kommando auf, das vom Betriebssystem ausgeführt
  - Chunk-Processing-Tasklet

Chunk-Verarbeitung



Eine Step-Definition unterscheidet:

- tasklet()
- chunk(chunkSize)
    - reader
    - processor
    - writer

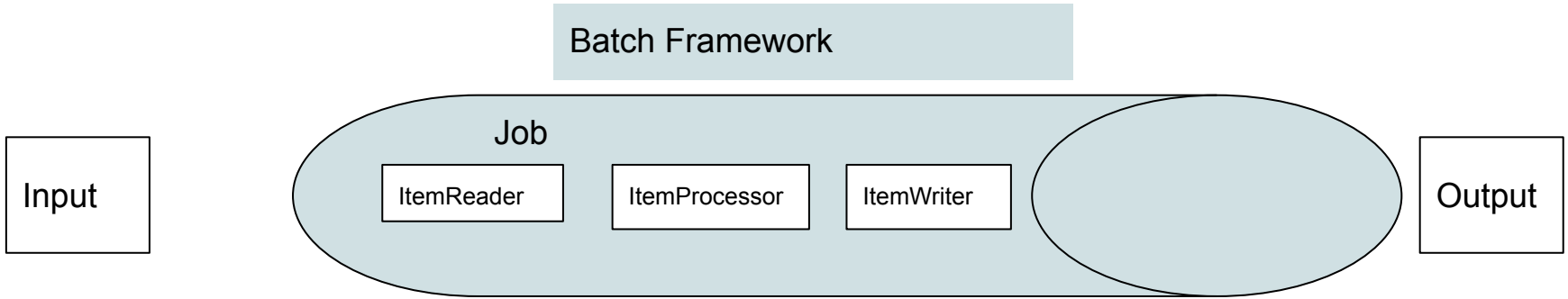
Programmier-Modell für Chunks

- Ausgerichtet auf Interfaces
  - interface ItemReader, ItemProcessor, ItemWriter
  - https://docs.spring.io/spring-batch/docs/current/api/index.html
- Custom Reader, Processor und Writer sind selbstverständlich möglich
  - die Interfaces sind für einen ersten Wurf sogar funktional
  - In der Praxis sind aber häufig nur Custom Processors notwendig
- Ausreichende Bibliothek von typischen Reader- und Writer-Implementierungen ist Bestandteil von Spring Batch
  - Im Unterschied dazu ist Java Batch nur eine Spezifikation

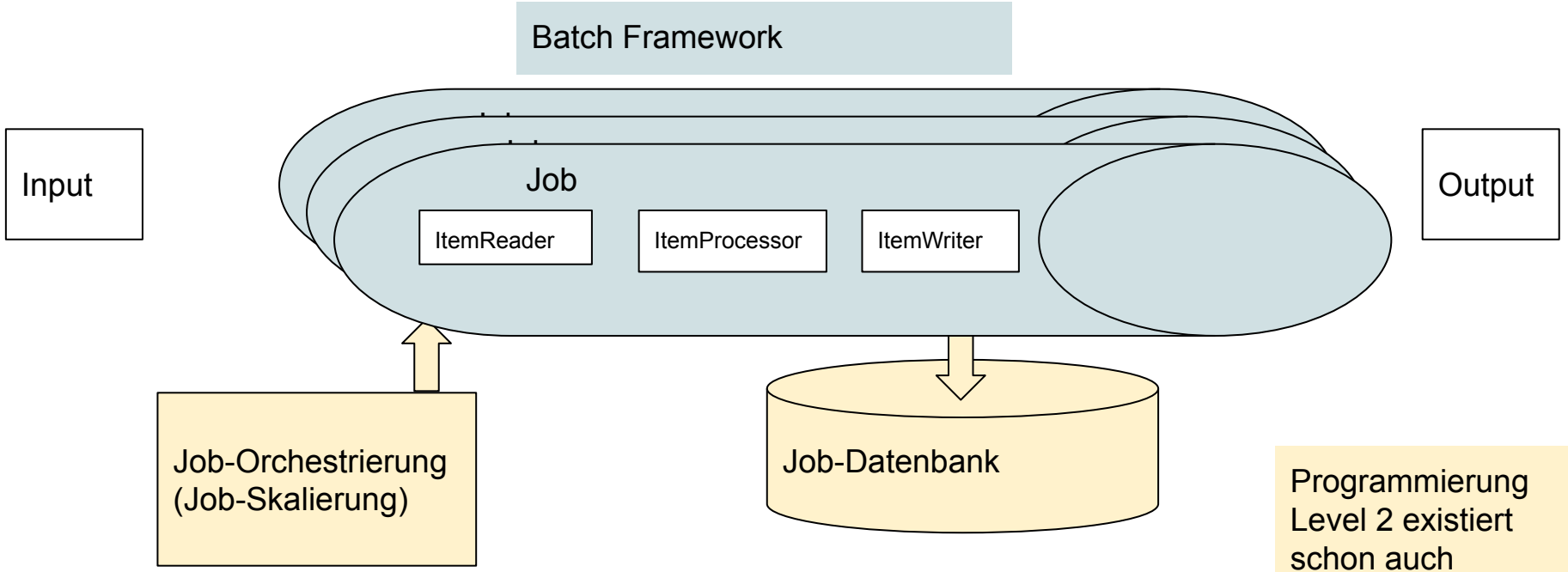
Todo

- Schreiben Sie eine komplett neue Job-Definition
  - @Configuration @EnableBatchProcessing
- Implementieren Sie einen einfachen Custom Reader/Processor/Writer
  - Reader liefert eine Menge von Namen aus einer vordefinierten Liste
    - Hinweis: return null ist das Kennzeichen für “keine weiteren Daten vorhanden”
  - Processor: Umwandlung der Zeichenkette in eine Zahl
    - string.length()
  - Writer
    - Konsolenausgabe der Chunks
  - Hinweis: Realisieren Sie diese als vollständige Klassen
- Testen
- Optional
  - MethodInvokingTaskletAdapter
    - Refactoring der Hello-World-Applikation in eine HelloWorld-Business-Klasse

Programmierung Level 1



Warum “Batch”

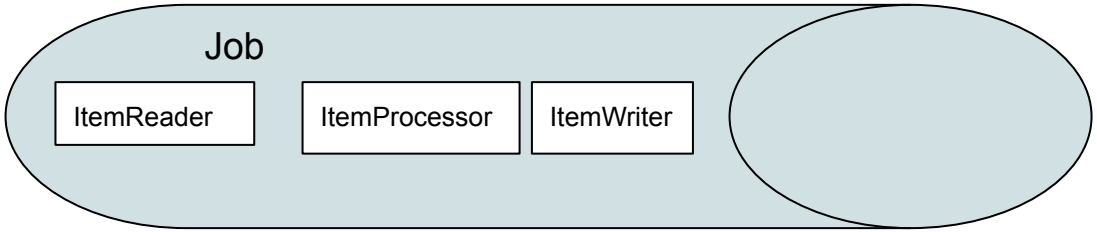

Batch Framework
Input
Job
ItemReader
ItemProcessor
ItemWriter
Output
Job-Orchestrierung
(Job-Skalierung)
Job-Datenbank
Programmierung
Level 2 existiert
schon auch

Fachliche Vorgabe für Level 1

Datei mit Eingangsdaten

Pro Zeile ein unstrukturierter Datensatz

z.B. eine Liste von Namen



Datei mit dem Ergebnis

Spring Batch

- FlatFileItemReader<T>
- FlatFileItemWriter<T>
- Beide sind massiv konfigurierbar
  - Zur Konfiguration wird in den meisten Fällen der korrespondierende Builder benutzt
- ToDo
  - Ersetzen Sie in Ihrer simpleChunk-Konfiguration den Reader und Writer!
  - Optional:
    - Arbeiten mit Jdbc-Reader oder Writer
    - Hinweis

Wie geht es weiter in Level 2?

integrata cegos

- Bisher nur ein einziger Step
- Nun:
  - Ein Flow von Steps
    - Steuerung des Flows
    - Fehlerbehandlung innerhalb eines Flows
  - Job- und Step-Lifecycle
    - @BeforeStep, @AfterStep
  - Validierung
    - Job Parameter
    - Fehlerhafte Datensätze

Exkurs: Spring

- Spring Batch unterstützt den so genannten “ExecutionContext” sowie den “StepContext”
- Spring selbst unterstützt den so genannten “Application Context” mit den zugehörigen “Scopes”
  - Lebensdauer einer Java-Instanz
    - Standard-mäßig: “ApplicationScope”
  - @StepScope
  - @JobScope

Umgang mit Fehlern beim Lesen von Daten

integrata cegos

- Step-Konfiguration
  - Jeder Step kann Fehler-tolerant sein
    - faultTolerant()
  - Bestimmte Exceptions können als “skip”-Kriterium genutzt werden
    - skip(MyException.class)
  - Maximal-Anzahl tolerabler Skips kann angegeben werden
    - skipLimit(2)
- Listener mit einer @OnReadError public handler(Exception e){

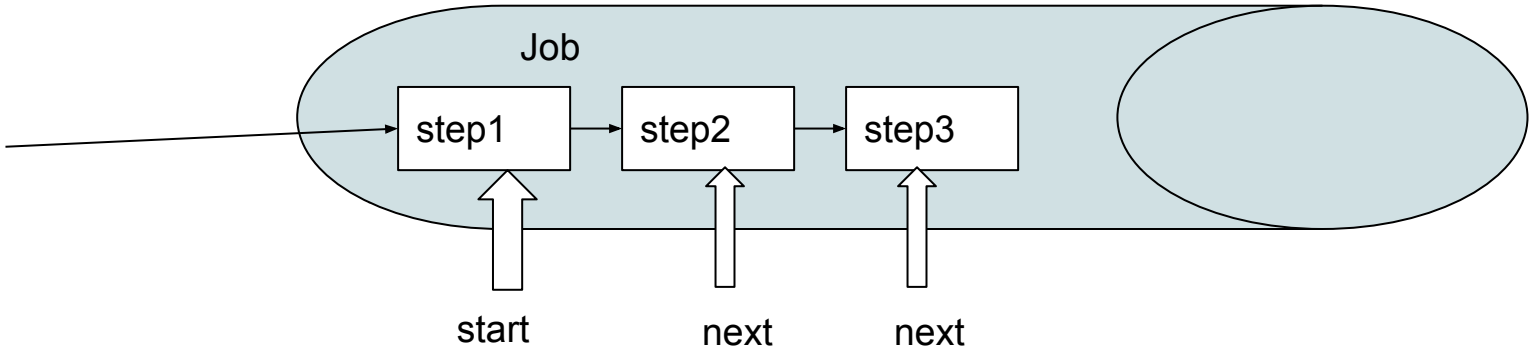
```
if (e instanceof FlatFileParseException){
    ((FlatFileParseException)e).getLineNumber()
}
}
```

restartability eines Jobs

- Konfiguration
  - preventRestart()

Flows

- Bisher
  - ein einzelner Step als “start”
- Nun
  - JobBuilder hat auch eine Funktion next(Step)



Flows

- Verzweigungen
  - Berücksichtigen den ExitStatus
- Beispiele: Job-Definition
  - start(firstStep).on(“FAILED”).to(failStep)
  - start(firstStep).on(“FAILED”).end()
  - start(firstStep).on(“FAILED”).to(failStep).from(firstStep).on(“SUCC*”).toSuccess(successStep)
- Es können auch eigene ExitStatus benutzt werden

Orchestrierung von Batch-Jobs