Treffpunkt „Semicolon“

Java-Batch

Der neue Standard in Java EE 7

Thomas Much

22.10.2013

Über...

- **Thomas Much**
- Dipl.-Inform. (Universität Karlsruhe (TH))
- Berater, Architekt, Entwickler, Coach (seit 1990)
- Trainer für Workshops / Seminare (seit 1998)
- Autor
- Speaker

BATCH ???

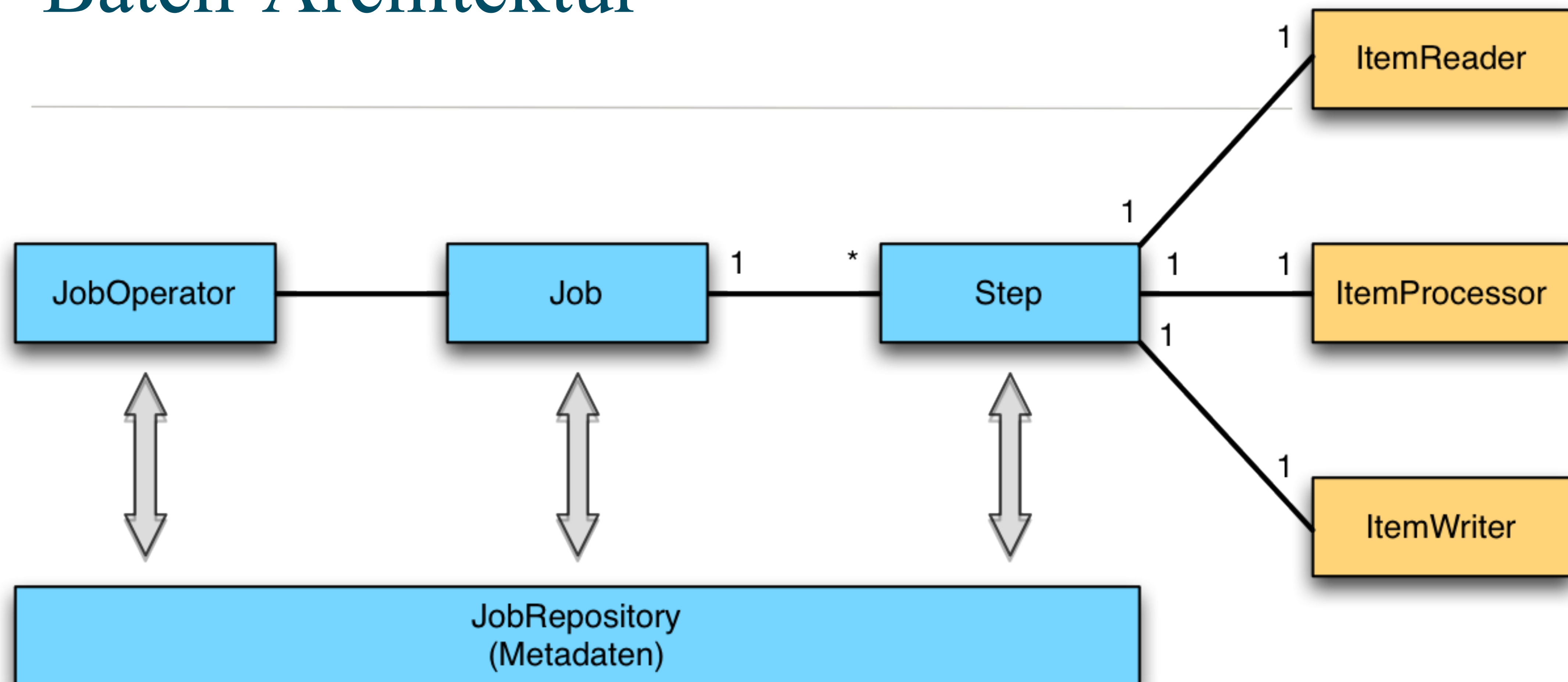
http://commons.wikimedia.org/wiki/File:Various_dinosaurs.png

- War das nicht irgend etwas aus dem *letzten Jahrtausend* auf dem HOST?

Was ist Batch-Verarbeitung?

- Regelmäßige, geschäftskritische Aufgaben
  - Abrechnungen, Archivierung, Mailversand, PDF-Generierung ...
- Zeit- oder Ereignis-gesteuert
  - z.B. am Tages-/Monatsende
- Oder auf Benutzer-Anforderung
  - Aber ohne Benutzer-*Interaktion*
- Typischerweise
  - Lang laufend (rechenintensiv) und/oder
  - Massendatenverarbeitung (datenintensiv)
- Übliche Features
  - Start/Stop, Checkpoints, Partitionierung, Parallelisierung

Batch-Architektur



- Seit Jahrzehnten typische Batch-Architektur
- Genutzt z.B. in COBOL, JCL, C, C++, C#, Java ...

- Datenquelle und -ziel der Schritte sind üblicherweise Datenbanken und/oder Dateien

Batch-Verarbeitung mit Java EE 7

- „Batch Applications for the Java Platform“ (JSR-352)
- http://jcp.org/en/jsr/detail?id=352
- Übernimmt viele Ideen aus Spring Batch, WebSphere Compute Grid u.a.

- Programmierung mit `javax.batch.**`
- Konfiguration mittels
  Job Specification Language (JSL) – „Job XML“

- Setzt Java 6 voraus
- Standalone und im Java-EE-Container nutzbar

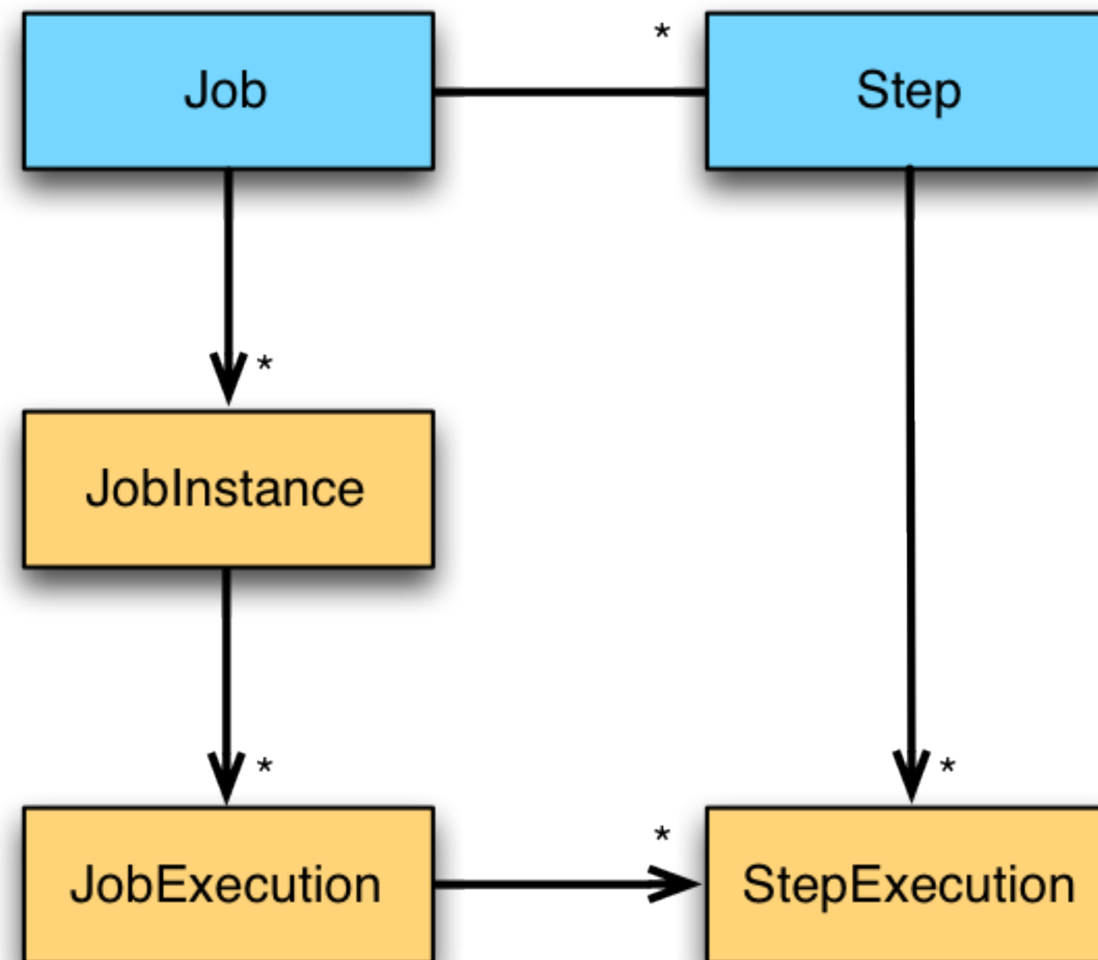
- Repository-Implementation kein Teil der Spezifikation

Jobs und Steps (1)

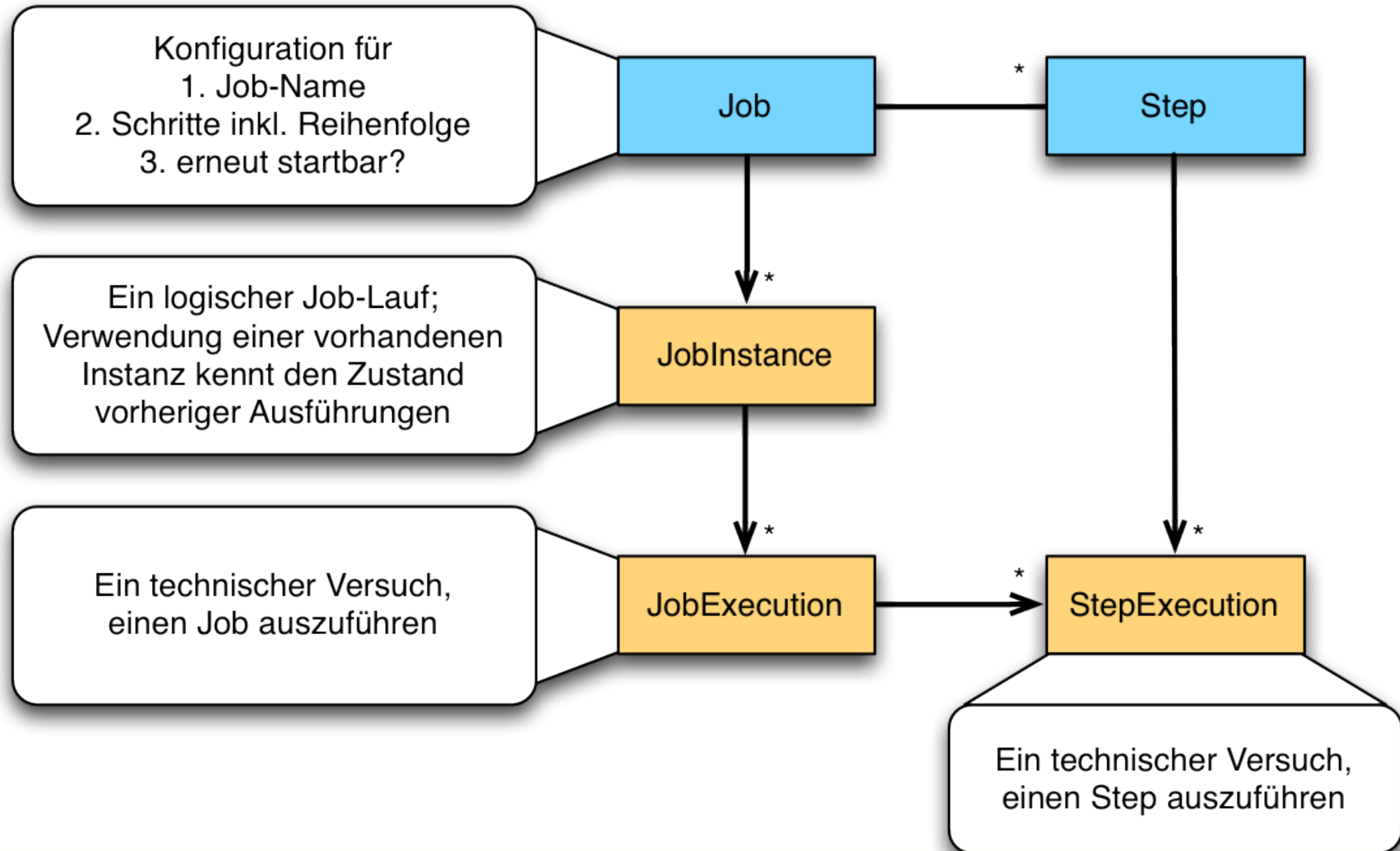
täglicher **Abrechnungs-Job**

Abrechnungs-Job **04.09.2013**

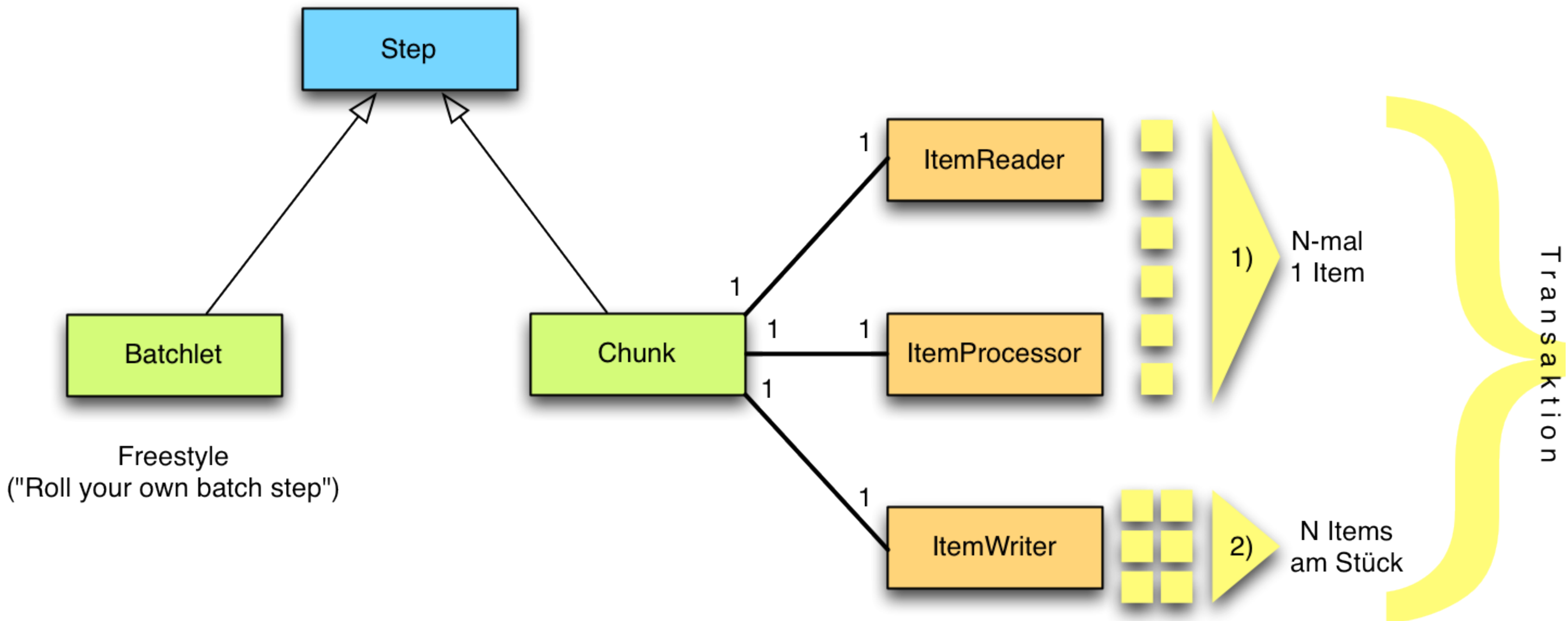
1. Versuch Abrechnungs-Job 04.09.2013



Jobs und Steps (2)



Steps – Chunks und Batchlets



Chunk-orientierte Batch-Verarbeitung

- „Chunk“ (Brocken/Batzen/Block)
- Standard-Vorgehen für Java-Batch-Schritte

- Für einen Chunk-Step sind Implementationen der folgenden Interfaces nötig (bzw. Unterklassen, sofern möglich):
  - `ItemReader` (bzw. `AbstractItemReader`)
  - `ItemProcessor`
  - `ItemWriter` (bzw. `AbstractItemWriter`)

- *Alle* Methoden deklarieren „`throws Exception`“

- Der Abschluss eines Chunks erzeugt einen Checkpoint

Chunks – ItemReader (1)

```
import javax.batch.api.chunk.*;

public class MeinItemReader implements ItemReader {

  public void open(Serializable checkpoint) { ... }

  public Object readItem() { ... }

  public Serializable checkpointInfo() { ... }

  public void close() { ... }
}
```

null beim ersten Aufruf innerhalb der Job-Instanz

null, wenn keine weiteren Daten vorhanden sind (→ Chunk-Commit und Step-Ende)

Wird vor dem Chunk-Commit aufgerufen

Chunks – ItemReader (2)

```
import javax.batch.api.chunk.*;

public class MeinItemReader implements ItemReader {

  private Integer nextId; // üblicherweise ein POJO

  public void open(Serializable checkpoint) {
    if (checkpoint == null) nextId = 0;
    else nextId = (Integer)checkpoint;
  }

  public Object readItem() { nextId++; ... }

  public Serializable checkpointInfo() {
    return nextId;
  }
}
```

Chunks – ItemProcessor

```
import javax.batch.api.chunk.*;

public class MeinItemProcessor
                       implements ItemProcessor {

  public Object processItem(Object item) {

    Bestellung b = (Bestellung)item;

    ...

    return new SummeDTO( b.getSumme() );
  }
}
```

Geht an den ItemWriter
(null → Item wird ausgefiltert)

Kommt vom ItemReader

Chunks – ItemWriter

```
import javax.batch.api.chunk.*;

public class MeinItemWriter implements ItemWriter {

  public void open(Serializable checkpoint) { ... }

  public void close() { ... }


  public void writeItems(List<Object> items) { ... }


  public Serializable checkpointInfo() { ... }

}
```

Writer-Checkpoint innerhalb der Job-Instanz; unabhängig vom Reader-Checkpoint!

Wird vor dem Chunk-Commit aufgerufen

Task-orientierte Verarbeitung (Batchlets)

```
import javax.batch.api.*;

public class MeinBatchlet implements Batchlet {

  public String process() { ... }

  public void stop() { ... }

}
```

Oder
`extends AbstractBatchlet`

Bel. Exit-Status

Gesamte Verarbeitungslogik des Batchlet-Steps.
Bei Exception hat der Step den Batch-Status `FAILED`.

Wird nach `JopOperator.stop()` ausgeführt
(auf einem anderen Thread als `process()`).
Sollte sinnvoll implementiert werden.

Job Specification Language („Job XML“)

META-INF/batch-jobs/meinJob.xml

```
<job id="meinJob">

  <step id="schritt1" next="schritt2">
    <chunk item-count="10">
      <reader ref="meinItemReader" />
      <processor ref="meinItemProcessor" />
      <writer ref="meinItemWriter" />
    </chunk>
  </step>

  <step id="schritt2">
    <batchlet ref="meinBatchlet" />
  </step>

</job>
```

id = Job-Name

optional!

Kein nächster Schritt → Ende

Konfiguration & Packaging

META-INF (JAR) bzw.
WEB-INF/classes/META-INF (WAR)

META-INF/batch-jobs/meinJob.xml

```
...
<reader ref="meinItemReader" />
...
```

Suchreihenfolge:
1. implementationsspezifisch (optional)
2. `batch.xml` (sofern vorhanden)
3. `ref`-Wert als Klassennamen behandeln

META-INF/batch.xml (optional)

```
<batch-artifacts>
  <ref id="meinItemReader"
       class="com.muchsoft.batch.MeinItemReader" />
  ...
</batch-artifacts>
```

Klasse muss einen Default-Konstruktor besitzen

Batch-Jobs ausführen

```
import javax.batch.operations.*;
import javax.batch.runtime.*;

...

JobOperator jobOp = BatchRuntime.getJobOperator();

Properties params = new Properties();

params.put("meinJobInputFile", "abrechnung.csv");

long jexecId = jobOp.start("meinJob", params);
```

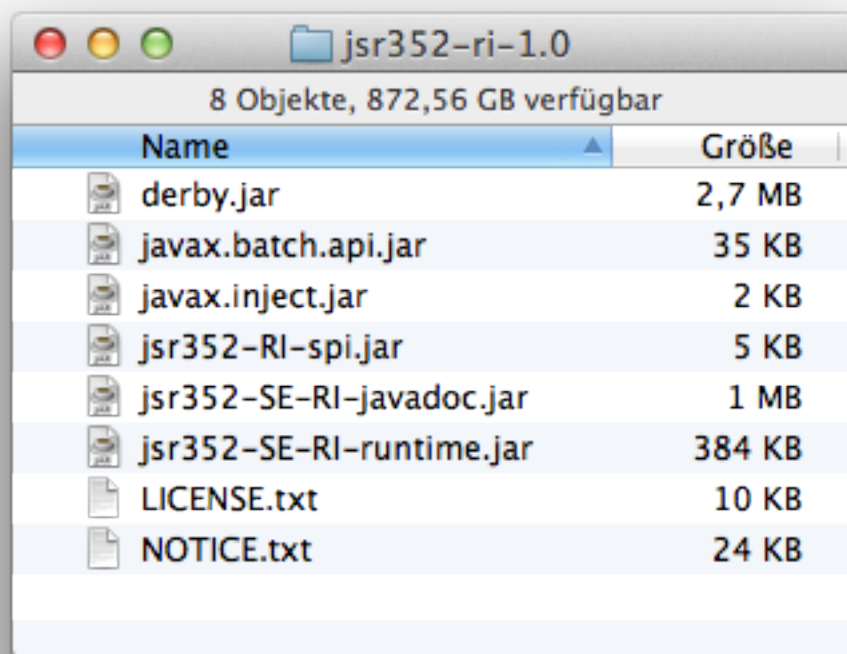
Kehrt sofort zurück.

Erzeugt neue JobInstance *und* neue JobExecution

META-INF/batch-jobs/meinJob.xml

Referenzimplementation

- „JBatch“
- Stammt von IBM
- https://java.net/projects/jbatch/

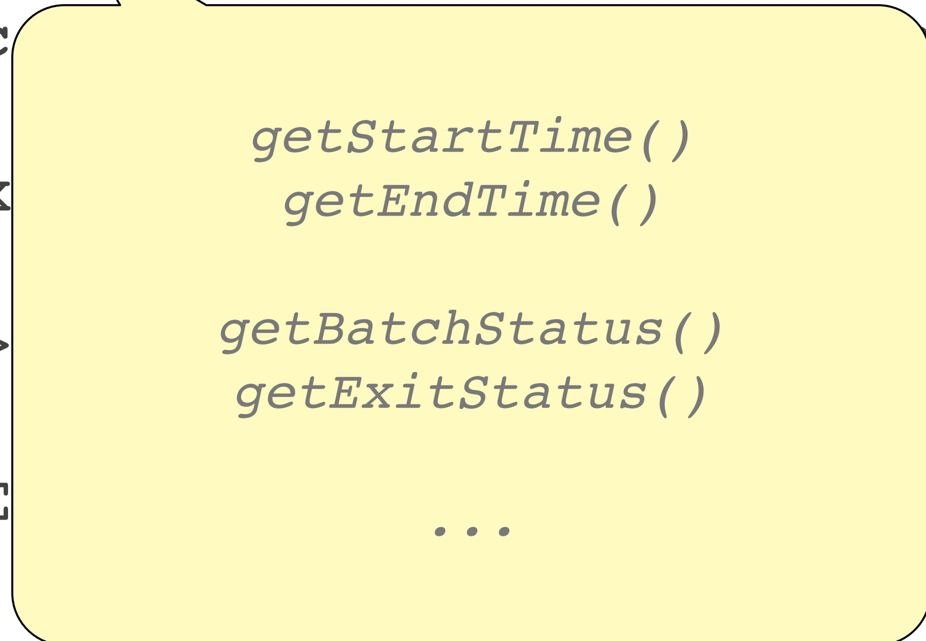


Batch-Jobs abfragen

```
JobOperator jobOp = BatchRuntime.getJobOperator();
...

JobExecution exec = jobOp.getJobExecution(jexecId);

JobInstance jinst = jobOp.getJobInstance(jexecId);

List<JobExecution> execs = jobOp.getJobExecutions(jinst);

List<Long> execIds = jobOp.getRunningExecutions("meinJob");

List<StepExecution> stepExecs =
                    jobOp.getStepExecutions(jexecId);

Properties execParams = jobOp.getParameters(jexecId);

Set<String> jobNames = jobOp.getJobNames();
```

Batch-Jobs abfragen – JobExecution

```
JobOperator jobOp = BatchRuntime.getJobOperator();
...

JobExecution exec = jobOp.getJobExecution(jexecId);

JobInstanc                                 stance(jexecId);

List<JobEx                                 etJobExecutions(jinst);

List<Long>                                 ingExecutions("meinJob");

List<StepE
                                           Executions(jexecId);

Properties execParams = jobOp.getParameters(jexecId);

Set<String> jobNames = jobOp.getJobNames();
```



Batch-Jobs abfragen – StepExecution (1)

```
JobOperator jobOp                                   ator();
...

JobExecution exec                                   execId);

JobInstance jinst                                   xecId);

List<JobExecution>                                  utions(jinst);

List<Long> execId                                   ions("meinJob");

List<StepExecution> stepExecs =
                         jobOp.getStepExecutions(jexecId);

Properties execParams = jobOp.getParameters(jexecId);

Set<String> jobNames = jobOp.getJobNames();
```

```
getStartTime()
getEndTime()

getBatchStatus()
getExitStatus()

getPersistentUserData()

getMetrics()

...
```

Batch-Jobs abfragen – StepExecution (2)

```
JobOperator jobOp                                       ator();
...

JobExecution exec

JobInstance jinst

List<JobExecution>

List<Long> execId

List<StepExecution> stepExecs =
                    jobOp.getStepExecutions(jexecId);

Properties execParams = jobOp.getParameters(jexecId);

Set<String> jobNames = jobOp.getJobNames();
```

getStartTime()
getEndTime()

getBatchStatus()
getExitStatus()

getPersistentUserData()

***getMetrics**(): Metric[]*

...

Enum Constant and Description
COMMIT_COUNT
FILTER_COUNT
PROCESS_SKIP_COUNT
READ_COUNT
READ_SKIP_COUNT
ROLLBACK_COUNT
WRITE_COUNT
WRITE_SKIP_COUNT

JobContext und StepContext (1)

```
import javax.batch.runtime.context.*;
import javax.inject.*;

public class MeinItemWriter implements ItemWriter {

   @Inject
   private JobContext jobContext;

   @Inject
   private StepContext stepContext;

   ...
}
```

Die Batch-Laufzeitumgebung muss sicherstellen, dass @Inject für JobContext und StepContext auch ohne weiteren Dependency-Injection-Container funktioniert!

- Batch-Kontexte sind an einen Thread gebunden.

JobContext und StepContext (2)

```
import javax.batch.runtime.context.*;
import javax.inject.*;

public class MeinItemWriter implements ItemWriter {

  @Inject
  private JobContext jobContext;

  @Inject
  private StepConte

  ...
}
```

```
getBatchStatus()
getExitStatus()
setExitStatus()

getTransientUserData()
setTransientUserData()

getProperties()

...
```

JobContext und StepContext (3)

```
import javax.batch.runtime.context.*;
import javax.inject.*;

public class MeinItemWriter implements ItemWriter {

  @Inject
  private JobContext jobContext;

  @Inject
  private StepContext stepContext

  ...
}
```

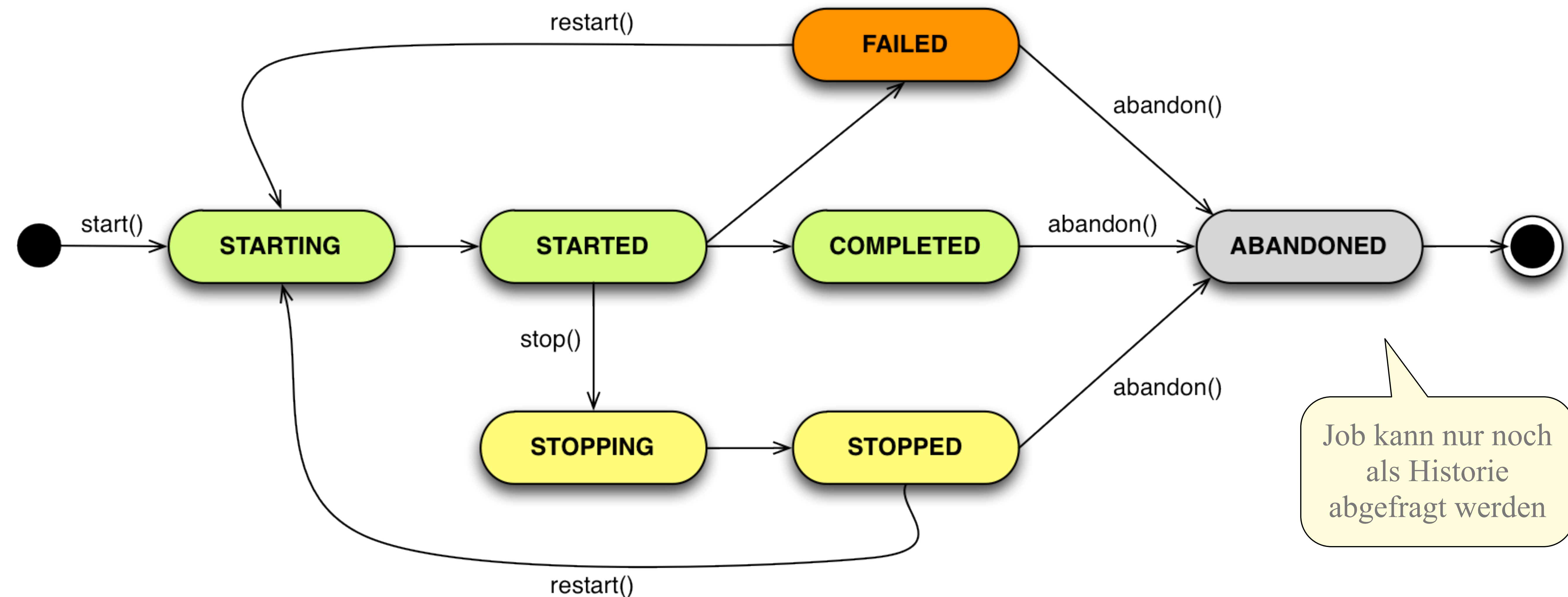
Batch-Status und Exit-Status

- *Jeder Step* endet mit einem Wert für den Batch-Status und den Exit-Status.
- Daraus wird ein *Gesamtstatus für den Job* ermittelt.

Enum Constant and Description
ABANDONED
COMPLETED
FAILED
STARTED
STARTING
STOPPED
STOPPING

- Batch-Status
  - Wird von der Batch-Runtime gesetzt
  - Enum-Wert
- Exit-Status
  - Wird durch die Anwendung oder durch Job XML gesetzt
  - Beliebiger String
  - Entspricht dem Batch-Status, solange kein Exit-Status explizit gesetzt wurde

Batch-Status und JobOperator-Methoden



Exceptions und Fehlerbehandlung (1)

- Generell führen *unbehandelte Exceptions* zum Job-Abbruch mit dem Batch-Status FAILED.

- Exceptions können behandelt werden, indem sie
  - *ignoriert* werden oder
  - die Chunk-Verarbeitung *wiederholt* wird.

- In der Job XML können *Transitionen* für einen Step-Exit-Status angegeben werden.
  - FAILED kann also einfach einen weiteren Step ansteuern.

Exceptions und Fehlerbehandlung (2)

```
<chunk skip-limit="…">
  <skippable-exception-classes>
    <include class="java.lang.Exception"/>
    <exclude class="java.io.FileNotFoundException"/>
  </skippable-exception-classes>
</chunk>
```

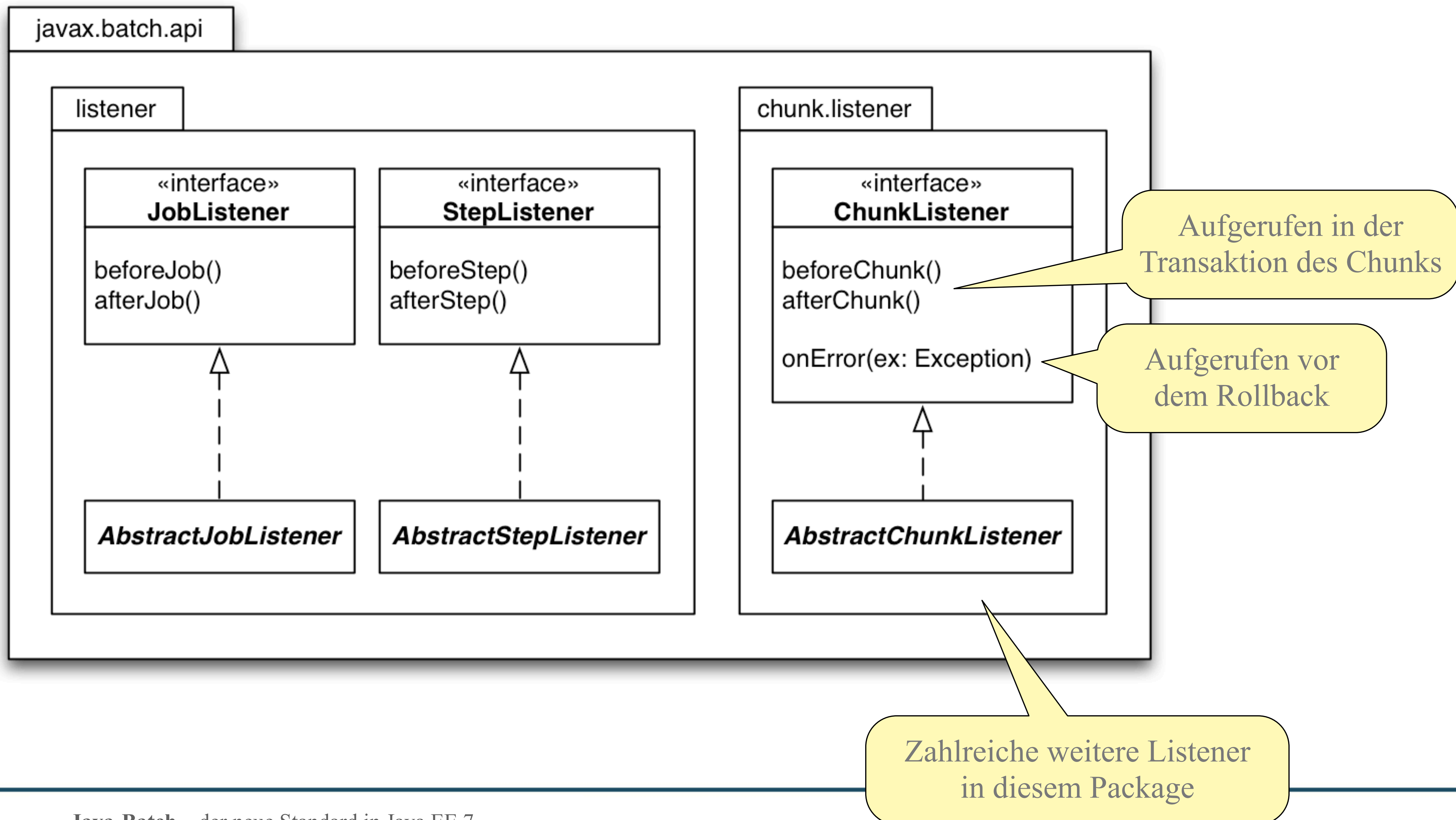
Default = kein Limit

```
<chunk retry-limit="…">
  <retryable-exception-classes>
    <include class="java.io.IOException"/>
    <exclude class="java.io.FileNotFoundException"/>
  </retryable-exception-classes>
</chunk>
```

Default = kein Limit

Eine Exception darf sowohl *retryable* als auch *skippable* sein!

Listener (1)



Listener (2)

- Weitere Listener-Interfaces mit abstrakter Impl.
  - `ItemReadListener`
  - `ItemProcessListener`
  - `ItemWriteListener`
- Listener für Exceptions
  - `SkipReadListener`
  - `SkipProcessListener`
  - `SkipWriteListener`

  - `RetryReadListener`
  - `RetryProcessListener`
  - `RetryWriteListener`

Aufruf erfolgt im selben Checkpoint-Scope wie der ItemReader/-Processor/-Writer, der die „Retryable Exception“ geworfen hat.

Eine Exception bricht den Job mit dem Batch-Status `FAILED` ab.

Listener – Konfiguration

```
<job id="meinJob">

  <listeners>
    <listener ref="..." />
    ...
  <listeners>

  <step id="...">
    <listeners>
      <listener ref="..." />
      ...
    <listeners>

    <chunk> ... </chunk>
  </step>

</job>
```

Nur Job-Listener.
Keine Aufruf-Reihenfolge festgelegt.

Bei Batchlets nur Step-Listener
(sonst alle außer Job-Listenern).
Keine Aufruf-Reihenfolge festgelegt.

Steps – Reihenfolge

- Erster <step> im <job> beginnt.
- next-*Attribut* im Step/Flow/Split kann Folge-Step/Flow/Split/Decision angeben:

```
<step id="schritt1" next="schritt2"> ... </step>
```

- Eine Schleife darf nicht definiert werden.
- Alternativ mit dem <next>-*Element*:

Exit-Status; Wildcards * und ? erlaubt

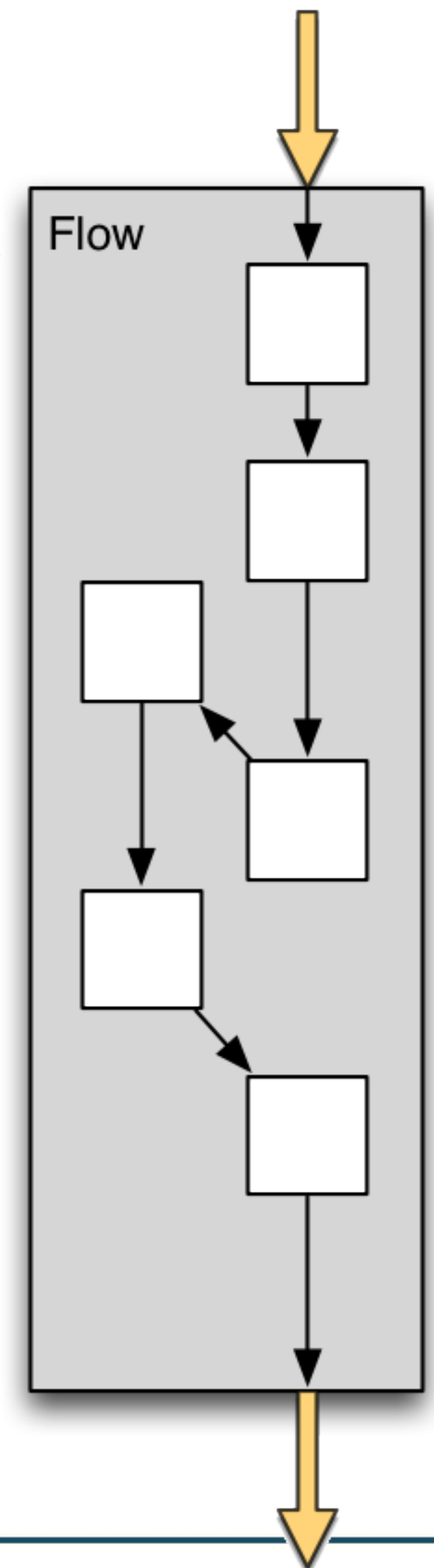
```
<step>
  <next on="COMPLETED" to="erfolgsMailSenden" />
  <next on="FAILED" to="fehlerMailSenden" />
</step>
```

Unbehandelter Exit-Status führt zu Batch-Status FAILED

Steps – Flow

- Sequenz von Step/Split/Decision/Flow
- Transitionen nur innerhalb des Flows erlaubt

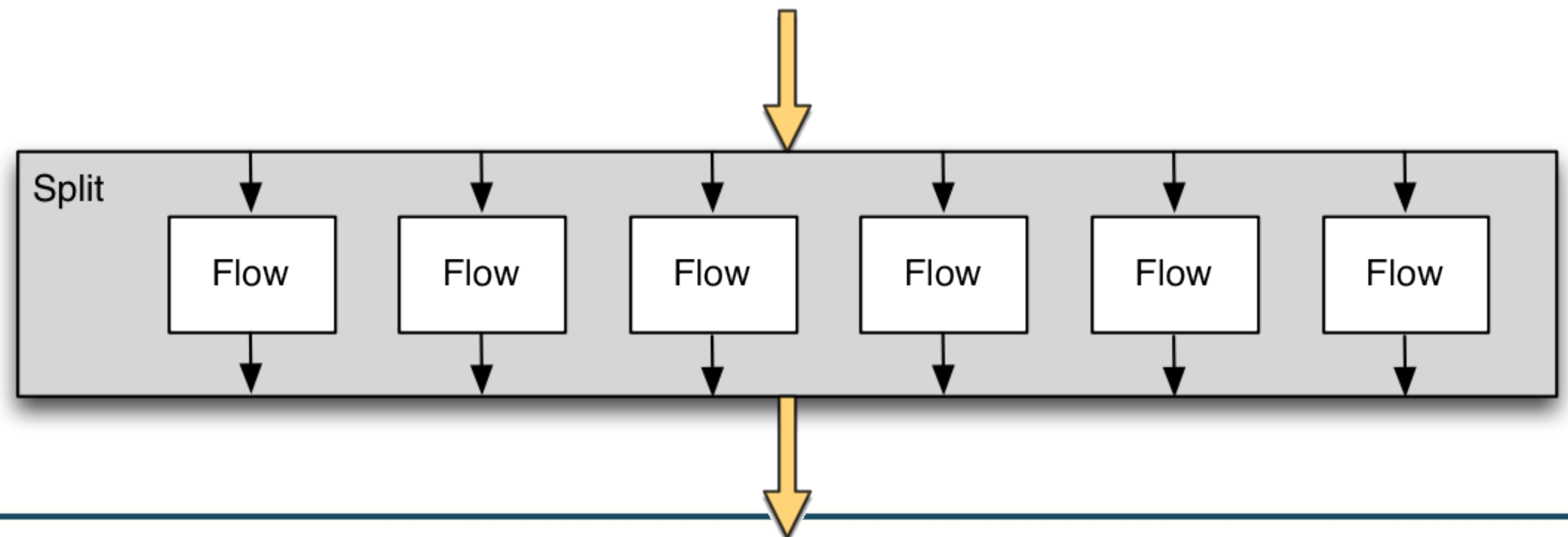
```
<job>
  <flow id="..." next="...">
    <step> ... </step>
    ...
  </flow>
</job>
```



Steps – Split

- *Parallelisierung* mit nebenläufigen Flows
- Jeder auf einem eigenen Thread

```
<job>
  <split id="..." next="...">
    <flow> ... </flow>
    <flow> ... </flow>
    ...
  </split>
</job>
```



Steps – Decision und Transitionen

```
<job>
  <step id="..." next="entscheidung">
  </step>

  <decision id="entscheidung" ref="meinDecider" >

    <next on="GoTo*" to="..." />
    <fail on="FAILED" exit-status="Fehler" />
    <end  on="COMPLETED" exit-status="Alles ok!" />
    <stop on="..." exit-status="..." restart="..." />

  </decision>
</job>
```

Nach Step/Flow/Split/Decision

```
import javax.batch.api.*;
public class MeinDecider implements Decider {
  public String decide(StepExecution[] executions) { ... }
}
```

Steps – Partitionierung (1)

- *Parallelisierung* durch Aufteilung der Items
- Jede Partition
  - läuft auf einem eigenen Thread
  - benötigt Properties, die die zu bearbeitenden Items festlegen
    - statisch per Job XML oder dynamisch per PartitionMapper

```
<step id="...">
  <chunk... /> oder <batchlet... />

  <partition>
    <plan partitions="2" threads="2">
      <properties partition="0"> ... </properties>
      <properties partition="1"> ... </properties>
    </plan>
  </partition>
</step>
```

Default = Anzahl d. Partitionen

Steps – Partitionierung (2)

```
<step id="...">
  <chunk... /> oder <batchlet... />

  <partition>
    <mapper ref="meinPartitionMapper" />
  </partition>
</step>
```

```
import javax.batch.api.partition.*;

public class MeinPartitionMapper implements PartitionMapper {
  public PartitionPlan mapPartitions() { ... }
}
```

Steps – Partitionierung (3)

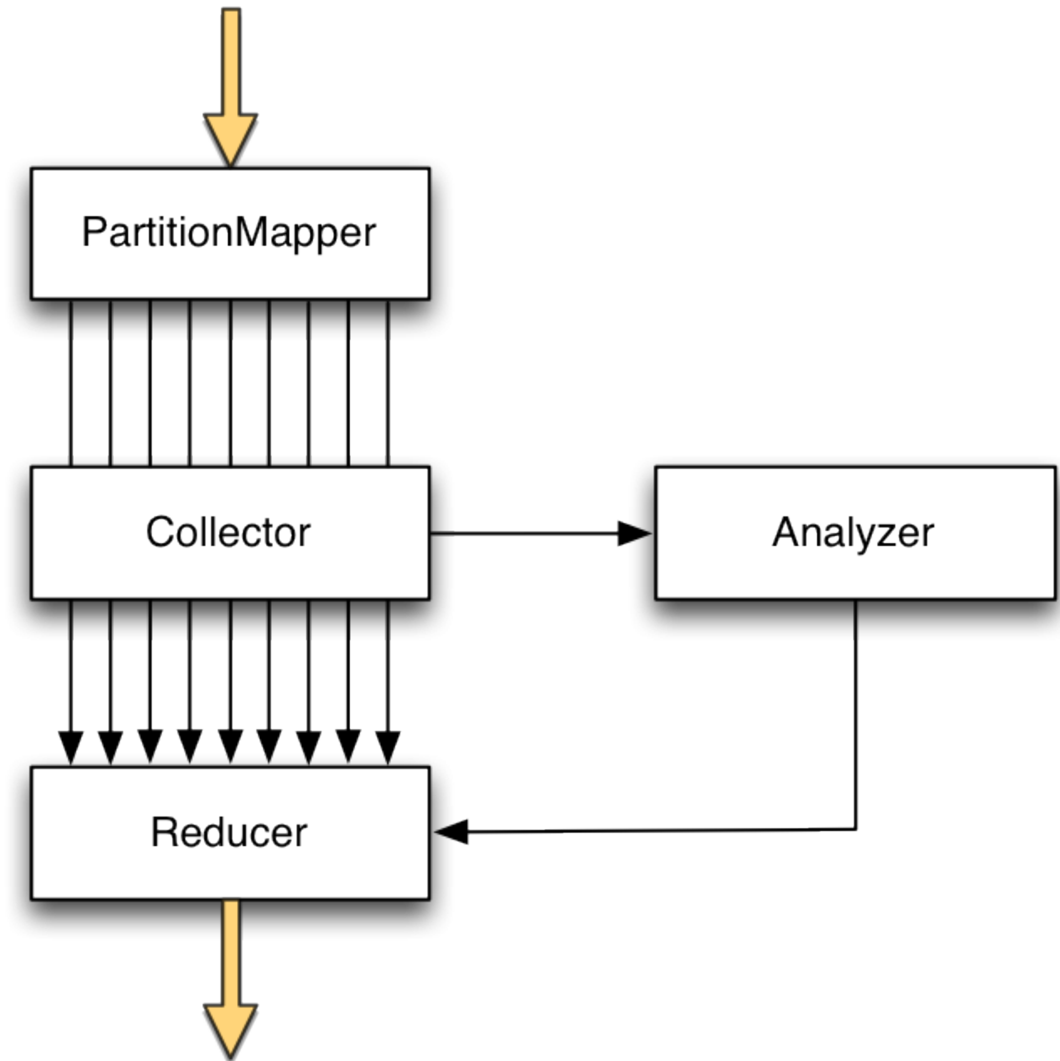
```
<step id="...">
  <chunk... /> oder <batchlet... />

  <partition>
    <mapper ref="meinPartitionMapper" />
  </partition>
</step>
```

```
getPartitions()
setPartitions(int)

getThreads()
setThreads(int)

getPartitionProperties()
setPartitionProperties(Properties[])
```

```
import javax.batc

public class MeinPartitionMapper implements PartitionMapper {
  public PartitionPlan mapPartitions() { ... }
}
```

Partitionierung – Map-Reduce

```
<step id="...">
  <chunk... /> oder <batchlet... />

  <partition>
    <mapper ref="..." />
    <reducer ref="..." />
    <collector ref="..." />
    <analyzer ref="..." />
  </partition>
</step>
```



Properties und Parameter

- Für alle Batch-Elemente (Job, Step, Chunk, ..., Batchlet, Listener, ...) können mittels Job XML statische *Properties* definiert werden:

```
<job id="meinJob">
  <properties>
    <property name="..." value="..." />
    ...
  </properties>
</job>
```

- Job-*Parameter* können dynamisch beim (Re)Start angegeben werden.
- Job- und Step-Properties können über die Kontexte abgefragt werden, Job-Parameter über den JobOperator.

@BatchProperty

- In alle Batch-Artefakte können Properties injiziert werden:

```
import javax.batch.api.*:
import javax.inject.*;

...

@Inject
@BatchProperty(name="meinJobInputFile")
private String csvInputFilename = "...";
```

Die Batch-Laufzeitumgebung muss sicherstellen, dass @Inject mit @BatchProperty auch ohne weiteren Dependency-Injection-Container funktioniert!

- Ist die Property nicht definiert, findet keine Injection statt (d.h. dann gilt der Java-Default-Wert).

Job XML – Attribut-Substitution (1)

- In Job XML kan *jeder* Attributwert nach folgendem Muster substituiert werden:

```
<job id="meinJob">
  <properties>
    <property name="basisname" value="abrechnung" />
  </properties>
  <step id="schritt1">
    <chunk>
      <properties>
        <property name="meinStepInputFile"
           value="#{jobProperties['basisname']}.csv" />
      </properties>
    </chunk>
  </step>
</job>
```

Job XML – Attribut-Substitution (2)

- Jede Substitutions-Property muss in einer der folgenden vier Kollektionen gesucht werden:
  - `jobParameters`
  - `jobProperties`
  - `systemProperties`
  - `partitionPlan`
- Wenn eine Property (noch) nicht definiert wurde, wird sie als Leerstring angenommen.
- Leerstrings können durch einen Default-Wert ersetzt werden:

```
"#{jobProperties['basisname']}?:abrechnung;.csv"
```

Benutzerdefinierte Checkpoints (1)

- Chunk-Checkpointing kann von einem einfachen Zähler auf einen selbst definierten Algorithmus umgestellt werden:

„Normales“ Checkpointing erhält man mit dem Default-Wert „item“ – nur dann werden auch die Attribute „item-count“ und „time-limit“ ausgewertet

```
<job id="meinJob">

  <step id="schritt1">

    <chunk checkpoint-policy="custom">

      <checkpoint-algorithm ref="meinCheckAlgo" />
      ...
    </chunk>

  </step>
</job>
```

Benutzerdefinierte Checkpoints (2)

```
import javax.batch.api.chunk.*;

public class MeinCheckpointAlgorithm
                implements CheckpointAlgorithm {
```

Wird für das nächste Checkpoint-Intervall abgefragt.
Angabe in Sekunden, 0 = kein Timeout.

```
  public int checkpointTimeout() { return 0; }
```

Wird nach der Verarbeitung *jedes* Items aufgerufen

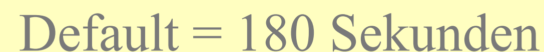
```
  public boolean isReadyToCheckpoint() { ... }

  public void beginCheckpoint() { ... }
  public void endCheckpoint() { ... }
}
```

Transaktionen

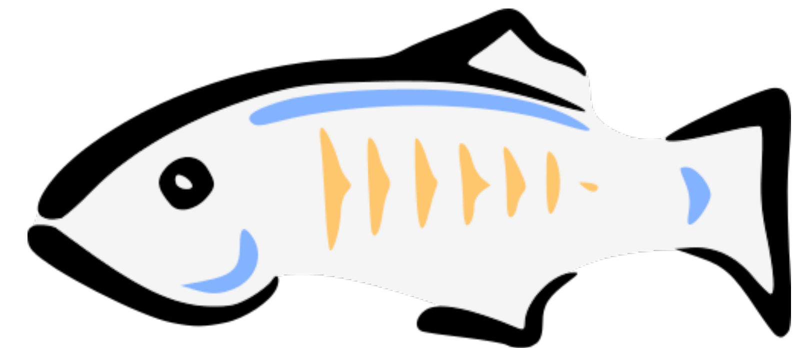
- In Java SE werden lokale Transaktionen verwendet,
- im Java-EE-Container globale JTA-Transaktionen.

- Transaktions-Timeout kann für Steps konfiguriert werden:

```
<step id="...">
  <properties>
    <property name="javax.transaction.global.timeout"
              value="600"/>
  </properties>
</step>
```



Batch-Anwendungen im Java-EE-Server



- Referenzimplementation „JBatch“ in GlassFish 4.0 enthalten
- http://glassfish.java.net/

- Unterstützung in JBoss/WildFly ab Version 8.0

- Integration mit Java EE 7
  - EJB, CDI, Timer, JPA, ...
  - Verwaltung der Batch-Jobs aber wie gehabt über BatchRuntime/JobOperator

GlassFish 4.0 – Admin Console (1)

GlassFish™ Server Open Source Edition

Tree

Common Tasks
- Domain
- server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
- Applications
- Lifecycle Modules
- Monitoring Data
- Resources
  - Concurrent Resources
  - Connectors
  - JDBC
  - JMS Resources
  - JNDI
  - JavaMail Sessions

General | Resources | Properties | Monitor | Batch | JMS Physical Destinations

Executions | Configuration

Batch Job Executions

View batch jobs that have been executed. Click an execution ID for details about a specific execution.

Instance Name: server

Batch Jobs (3)

Execution ID	Job Name	Batch Status	Exit Status	Instance ID	Start Time	End Time
37	erfolgreicherJob	COMPLETED	Alles ok!	37	2013-09-02 15:42:45 MESZ	2013-09-02 15:42:45 MESZ
36	erfolgreicherJob	COMPLETED	COMPLETED	36	2013-09-02 15:42:44 MESZ	2013-09-02 15:42:44 MESZ
38	fehlerhafterJob	FAILED	FAILED	38	2013-09-02 15:42:47 MESZ	2013-09-02 15:42:47 MESZ

GlassFish 4.0 – Admin Console (2)

GlassFish™ Server Open Source Edition

Tree

Common Tasks
- Domain
- server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
- Applications
- Lifecycle Modules
- Monitoring Data
- Resources
  - Concurrent Resources
  - Connectors
  - JDBC
  - JMS Resources
  - JNDI
  - JavaMail Sessions
  - Resource Adapter Configs
- Configurations

Execution Details | Execution Steps

Batch Job Execution Details

Back

View details about a specific batch job execution.

Instance Name: server

Job Name: erfolgreicherJob
Execution ID: 37
Step Count: 2
Batch Status: COMPLETED
Exit Status: Alles ok!
Start Time: 2013-09-02 15:42:45 MESZ
End Time: 2013-09-02 15:42:45 MESZ

Job Parameters (1)

Key	Value
eigenerExitStatus	true

GlassFish 4.0 – Admin Console (3)

GlassFish™ Server Open Source Edition

Tree

Common Tasks
- Domain
- server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
- Applications
- Lifecycle Modules
- Monitoring Data
- Resources
  - Concurrent Resources
  - Connectors
  - JDBC
  - JMS Resources
  - JNDI
  - JavaMail Sessions
  - Resource Adapter Configs
- Configurations
  - default-config
  - server-config
- Update Tool

Execution Details | Execution Steps

Batch Job Execution Steps

Back

View details about steps in a specific batch job execution.

Instance Name: server

Job Name: erfolgreicherJob

Execution ID: 37

JobSteps (2)

Step Name	Batch Status	Exit Status	Start Time	End Time	Step Metrics
schritt1	COMPLETED	COMPLETED	2013-09-02 15:42:45 MESZ	2013-09-02 15:42:45 MESZ	FILTER_COUNT 0 ROLLBACK_COUNT 0 READ_COUNT 0 COMMIT_COUNT 1 READ_SKIP_COUNT 0 WRITE_COUNT 0 PROCESS_SKIP_COUNT 0 WRITE_SKIP_COUNT 0
schritt2	COMPLETED	COMPLETED	2013-09-02 15:42:45 MESZ	2013-09-02 15:42:45 MESZ	FILTER_COUNT 0 ROLLBACK_COUNT 0 READ_COUNT 0 COMMIT_COUNT 1

GlassFish 4.0 – Admin Console (4)

GlassFish™ Server Open Source Edition

Tree

Common Tasks
- Domain
- server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
- Applications
- Lifecycle Modules
- Monitoring Data
- Resources
  - Concurrent Resources
  - Connectors
  - JDBC
  - JMS Resources
  - JNDI

General | Resources | Properties | Monitor | Batch | JMS Physical Destinations

Executions | Configuration

Batch Runtime Configuration

Save

Configure the batch runtime. The runtime uses a managed executor service and a data source to execute batch jobs.

Instance Name: server

Executor Service Lookup Name: concurrent/__defaultManagedExecutorService

JNDI lookup name of the managed executor service that provides threads to jobs

Data Source Lookup Name: jdbc/__TimerPool

JNDI lookup name of the data source that stores job information

GlassFish 4.0 – CLI (asadmin)

```
much$ ./asadmin list-batch-jobs -l
JOBNAME          APPNAME        INSTANCECOUNT  INSTANCEID  EXECUTIONID  BATCHSTATUS  STARTTIME                     ENDTIME
erfolgreicherJob Java-Batch-EE  2              37          37           COMPLETED    Mon Sep 02 15:42:45 CEST 2013 Mon Sep 02 15:
erfolgreicherJob Java-Batch-EE  2              36          36           COMPLETED    Mon Sep 02 15:42:44 CEST 2013 Mon Sep 02 15:
fehlerhafterJob  Java-Batch-EE  1              38          38           FAILED       Mon Sep 02 15:42:47 CEST 2013 Mon Sep 02 15:
```

```
much$ ./asadmin list-batch-job-executions -l
JOBNAME           EXECUTIONID  STARTTIME                      ENDTIME                        BATCHSTATUS  EXITSTATUS  JOBPARAMETER
 erfolgreicherJob 37           Mon Sep 02 15:42:45 CEST 2013  Mon Sep 02 15:42:45 CEST 2013  COMPLETED    Alles ok!   KEY
                                                                                                                      eigenerExitS
 erfolgreicherJob 36           Mon Sep 02 15:42:44 CEST 2013  Mon Sep 02 15:42:44 CEST 2013  COMPLETED    COMPLETED   KEY  VALUE
 fehlerhafterJob  38           Mon Sep 02 15:42:47 CEST 2013  Mon Sep 02 15:42:47 CEST 2013  FAILED       FAILED      KEY  VALUE
```

```
much$ ./asadmin list-batch-job-steps -l 37
STEPNAME  STEPID  STARTTIME                      ENDTIME                        BATCHSTATUS  EXITSTATUS  STEPMETRICS
schritt1  40      Mon Sep 02 15:42:45 CEST 2013  Mon Sep 02 15:42:45 CEST 2013  COMPLETED    COMPLETED   METRICNAME          VALUE
                                                                                                         READ_COUNT          0
                                                                                                         WRITE_COUNT         0
                                                                                                         COMMIT_COUNT        1
                                                                                                         ROLLBACK_COUNT      0
                                                                                                         READ_SKIP_COUNT     0
                                                                                                         PROCESS_SKIP_COUNT  0
                                                                                                         FILTER_COUNT        0
                                                                                                         WRITE_SKIP_COUNT    0
schritt2  41      Mon Sep 02 15:42:45 CEST 2013  Mon Sep 02 15:42:45 CEST 2013  COMPLETED    COMPLETED   METRICNAME          VALUE
                                                                                                         READ_COUNT          0
                                                                                                         WRITE_COUNT         0
                                                                                                         COMMIT_COUNT        1
                                                                                                         ROLLBACK_COUNT      0
                                                                                                         READ_SKIP_COUNT     0
                                                                                                         PROCESS_SKIP_COUNT  0
                                                                                                         FILTER_COUNT        0
                                                                                                         WRITE_SKIP_COUNT    0
```

Und was ist mit Spring Batch?

- Spring-Entwickler haben aktiv am offiziellen Standard mitgearbeitet.

- Nahezu gleiche Typ-Namen und -Funktionalitäten
- Kleine Unterschiede bei den (Job-)Scopes, bei der Chunk-Ausführung, bei den Properties ...
- XML-Konfiguration sehr ähnlich

- Spring Batch 3.0 wird JSR-352 implementieren
- Gerade als Milestone 1 erschienen
- http://blog.springsource.org/2013/08/23/spring-batch-3-0-milestone-1-released/

Was fehlt beim JSR-352 (1.0)?

- Implementationen von Jobs/Steps/Chunk-Elementen für typische Aufgaben
  - Der Standard definiert nur Interfaces und ein paar abstrakte Klassen
- Vorgaben bzgl. Skalierung über JVM-Grenzen hinweg
- Vorgaben bzgl. Thread-Konfiguration

Batch-Verarbeitung mit Java EE 7: Fazit (1)

- Umgebung und Programmiersprache haben sich geändert.
- Massendaten müssen aber nach wie vor verarbeitet werden.

- Der neue JSR-352-Standard hat nicht das Rad neu erfunden, sondern viel von den bisherigen De-Facto-Standards übernommen, kombiniert und angepasst – setzt damit also auf etablierte Best Practices.

http://www.speicherstadtmuseum.de/

Batch-Verarbeitung mit Java EE 7: Fazit (2)

- Sinnvoller Standard & sinnvolle Erweiterung von Java EE, da auch die Batch-Verarbeitung mehr und mehr in den Zuständigkeitsbereich von Java-Entwicklern fällt

- Praxiserfahrung fehlt
- dürfte aber keine bösen Überraschungen bieten

- Gute erste Version des Standards – wie immer mit Raum für Erweiterungen ☺

Treffpunkt „Semicolon“

Vielen Dank!

Thomas Much
info@muchsoft.com
www.muchsoft.com