integrata
cegos

Batch Frameworks für Java

Literatur

Spring Batch - Reference Documentation

Batch Applications for
the Java Platform

Einige Hinweise

- Die in diesem Seminar verwendete Werkzeuge und Frameworks sind Open Source
  - LPGL Lizenzmodell
- Dies ist ein Programmier-Seminar
  - Damit werden die Inhalte durch Übungen vertieft und verinnerlicht
  - Musterbeispiele werden zur Verfügung gestellt
  - Diese können am Ende des Seminars als ZIP-Datei kopiert werden
    - USB-Stick oder ähnliches
- Dokumentation und Ressourcen stehen auch im Internet zur Verfügung
- Konventionen
  - Befehle werden in `Courier-Schriftart` dargestellt
  - Dateinamen werden in *`kursiver Courier-Schriftart`* dargestellt
  - Links werden in `unterstrichener Courier-Schriftart` dargestellt

Copyright und Impressum

Javacream

Dr. Rainer Sawitzki

Alois-Gilg-Weg 6
81373 München

Inhalt

1

JAVA BATCH

1.1

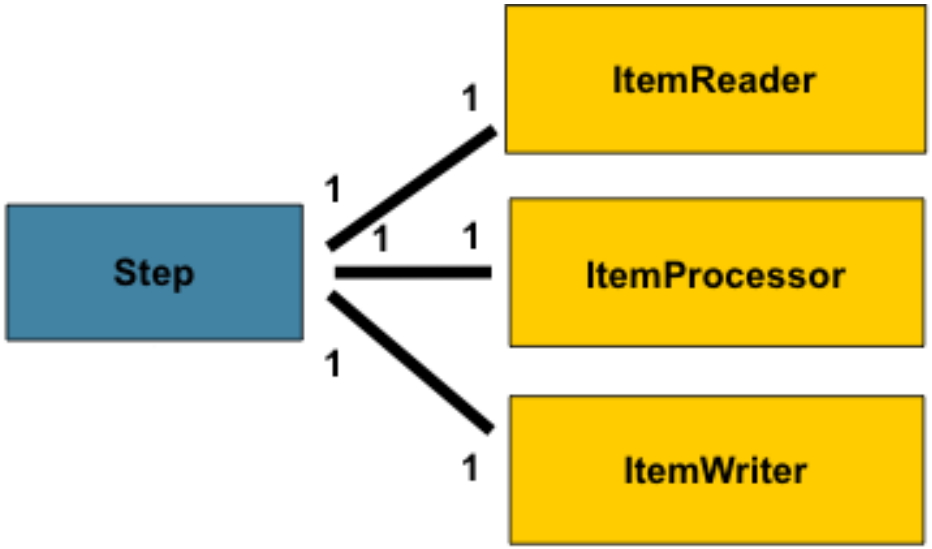
MERKMALE EINES BATCH-PROZESSES

Kennzeichen

- Kein Benutzer-Interaktion notwendig
  - Scheduled Ausführung
  - Event- oder Message-gesteuert
- Massendatenverarbeitung
- Typischerweise Parallelisierbar
  - Multithreading
  - Multi-Processing
  - Partitionierung der Eingangsdaten
- Automatisierte Fehlerbehandlung
  - Retry-Mechanismen
- Ausführliche Logging- und Monitoring-Werkzeuge
  - Laufende Jobs
  - Erfolgreiche und abgebrochene Läufe
  - Nachvollziehbare Fehlermeldungen

Abstraktion des Batch-Laufs

- Durch das Fehlen einer Benutzer-Interaktion lassen sich Batches extrem gut abstrahieren
  - Ein Batch-Lauf führt einzelne `Step`s aus
  - `Step`-Sequenz
    - Ein `ItemReader` liefert Eingabedaten
    - Diese werden von einem `ItemProcessor` verarbeitet
    - Die Ergebnisse werden von einem `ItemWriter` geschrieben



Typische Aufgaben eines Batches

- Conversion
  - Umwandlung der Eingabe-Daten in ein normalisiertes Format
- Validation
- Extract
  - Selektion aus einer Datenquelle in eine Ausgabe schreiben
- Extract/Update
- Processing and Upating
- Output/Format
- Basic Application Shell für Business Logic

Typische Optionen für den Job-Ablauf

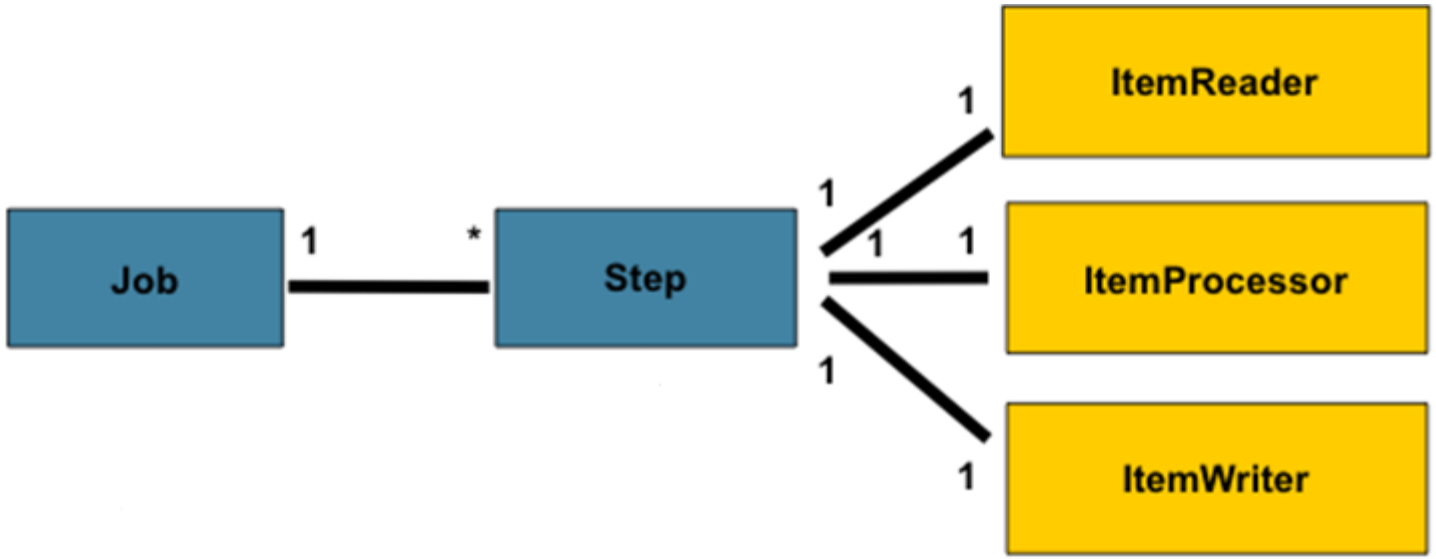
- Normal Processing, Single Process
  - Single commit am Ende des Laufs
- Konkurrierende Batch-Läufe
  - Locking-Mechanismen notwendig (optimistic, pessimistic)
- Parallel Processing
- Partitionierung
  - Parallelisierung auf partitionierten Datenbeständen

1.2

JOBS

Der Job

- Ein Job besteht aus mehreren aufeinanderfolgenden Steps

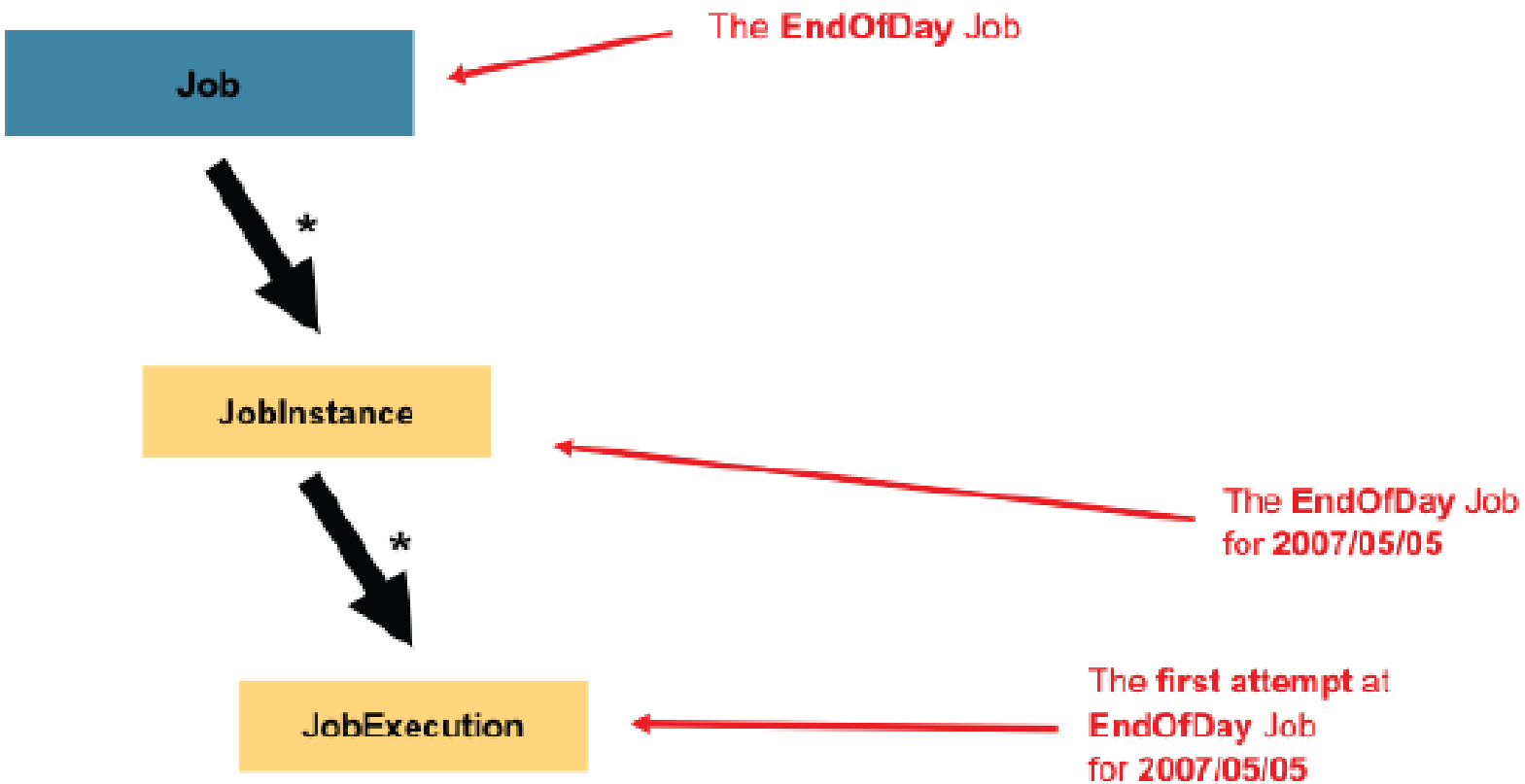


Job-Instanzen

- Ein Job kann wir eine Klasse instanziiert werden

Job Executions

- Eine Job-Instanz kann potenziell mehrfach ausgeführt werden
  - Jede Job-Instanz aggregiert Job-Executions

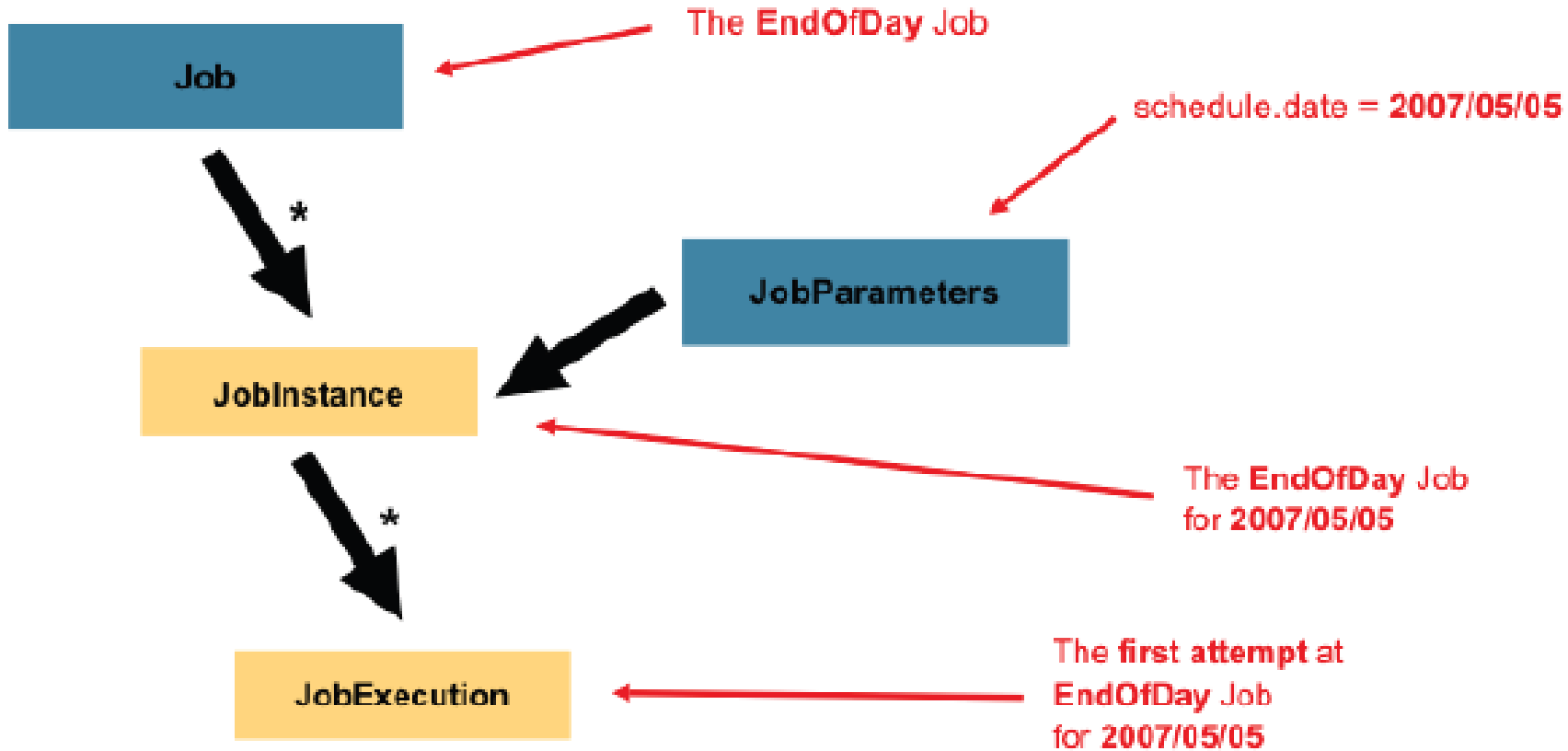


JobExecution: Eigenschaften

- `status`
- `start`
- `end`
- `exitStatus`
- `createTime`
- `lastUpdated`
- `executionContext`
- `failureExceptions`

Job Parameter

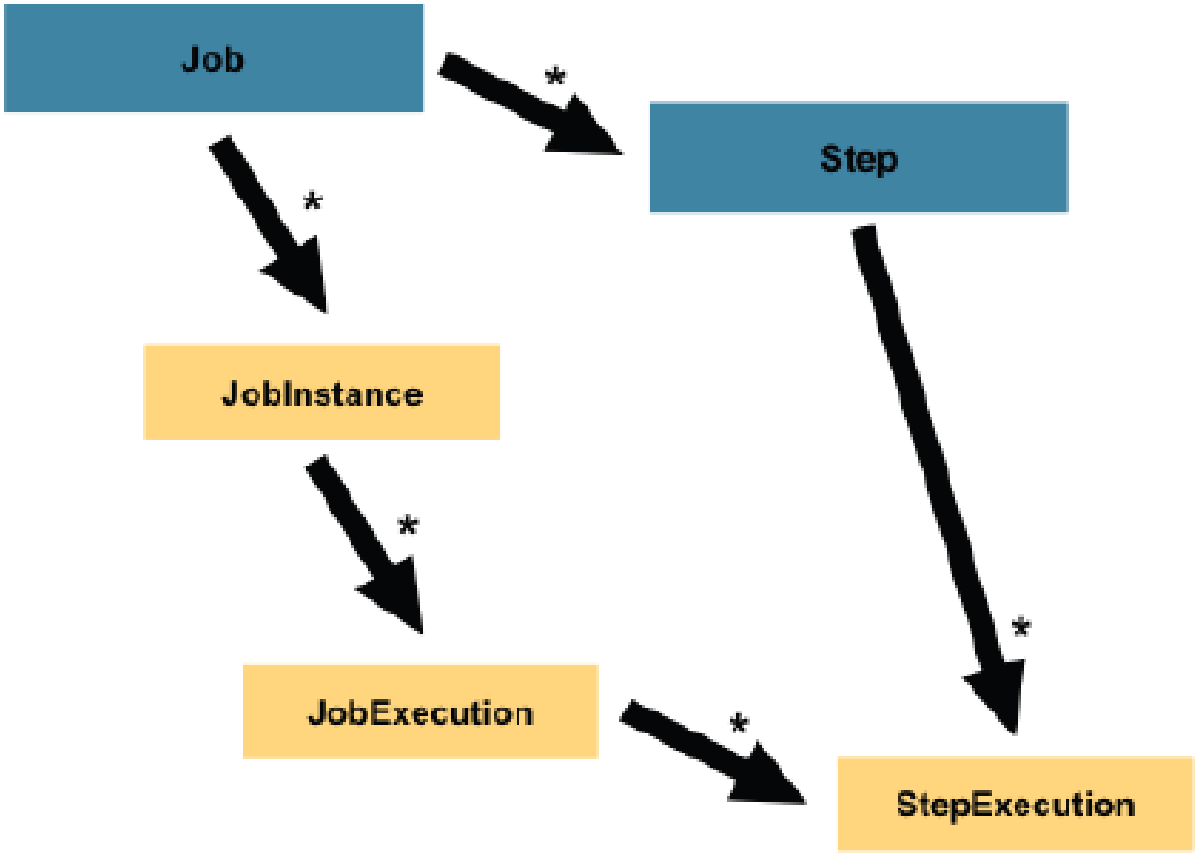
- Bei der Instanziierung eines Jobs werden JobParameter angegeben



1.3

STEPS

StepExecution



StepExecution: Eigenschaften

- `status`
- `startTime`
- `endTime`
- `exitStatus`
- `executionContext`
- `readCount`
- `writeCount`
- `commitCount`
- `rollbackCount`
- `readSkipCount`
- `processSkipCount`
- `filterCount`
- `writeSkipCount`

Aufgaben

- Step
  - Validierung
  - Formatierung
  - Sortieren
  - Aufsplitten
  - Merge
- Job
  - Parallelisierung
  - Zusammenführen

1.4

READER, PROCESSOR, WRITER

Typische ItemReaders

- File-driven
- Database-driven
- Message-driven
  - Einsatz eines Messaging-Systems notwendig
  - Aber auch
    - Remote Aufruf
    - Eingang einer Mail
    - …

2

BATCH SPEZIFIKATION

2.1

SPRING BATCH UND JSR 352

Spring Batch

- Subprojekt der Spring-Community
  - Mit der üblich hohen Qualität an Ressourcen
    - Dokumentation
    - Community
    - Kommerzieller Support über VMWare
- Komplette integriert in die übrige Spring-Infrastruktur
  - Spring Core als Dependency Injection und AOP-Framework
  - Einfache Benutzung zusätzlicher Module
    - Spring Integration
    - Spring Data
- Stand 2014: Version: 3.x

JSR 352

- Als Java Specification Request eine Spezifikation
  - Damit ein einheitliches API
    - Unter starkem Einfluss von Spring Batch definiert
      - Teile der Dokumentation sind identisch
  - Unabhängige Provider-Auswahl
    - Spring Batch ist ein Provider des JSR 352
      - Die Unterschiede zwischen Spring Batch und JSR 352 liegen im Detail
- Spezifikation in der Version 1.0 final seit 2013
  - Maintenance Release 2014

Referenz-Implementierung und Provider

- JBatch
  - Verfügbar für die Java Standard-Edition
  - Auch als JEE-Anwendung verfügbar
- Bestandteil der JEE
  - Seit der Version 6
  - Damit ist jeder Applikationsserver ein Batch-Provider
  - Der Batch-Prozess hat Zugriff auf die Infrastruktur des Applikationsservers

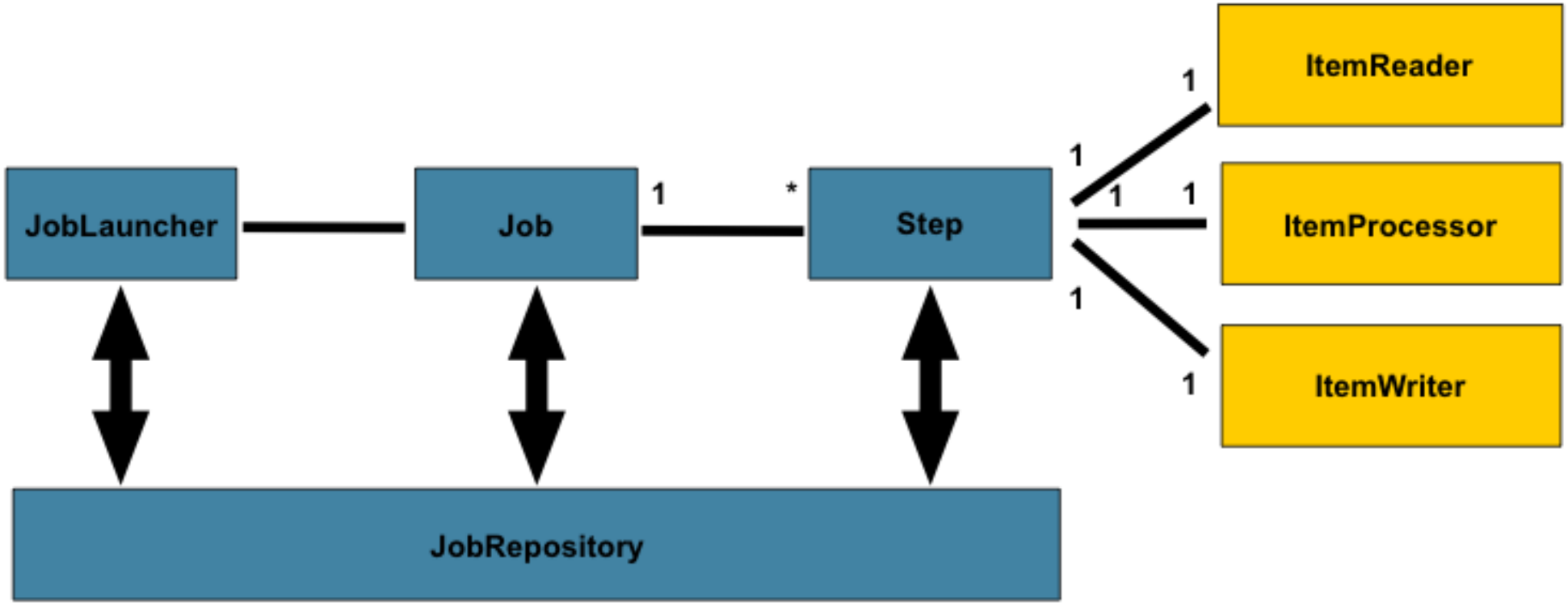
2.2

LAUFZEITUMGEBUNG

JobLauncher und JobRepository

- Der JobLauncher ist verantwortlich für das Starten von Jobs
- Das JobRepository enthält die Informationen zu laufenden Jobs
  - Zustand
  - JobParameter
  - Job-Executions

Batch Architektur



3

PROGRAMMIERUNG

3.1

JOB SPECIFICATION LANGUAGE

Job-Definition

- Jobs werden im XML-Format definiert
  - Ein Job ist Graphen-orientiert aufgebaut
  - Durchaus Ähnlichkeiten mit einem Business Process!

```
<job id="footballJob">
  <step id="playerload" next="gameLoad"/>
  <step id="gameLoad" next="playerSummarization"/>
  <step id="playerSummarization"/>
</job>
```

3.2

SPRING BATCH API

Spring Batch: `batch`-Namespace

integrata cegos

```
<beans:beans
xmlns="http://www.springframework.org/schema/batch"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/batch
http://www.springframework.org/schema/batch/spring-batch-
2.2.xsd">
<job id="ioSampleJob">
<step id="step1">
<tasklet>
<chunk reader="itemReader" writer="itemWriter" commit-
interval="2"/>
</tasklet>
</step>
</job>
</beans:beans>
```

Spring Batch: Fachklassen und Infrastruktur

integrata cegos

- Die Job-Definition ist eine normale Spring-Konfiguration
- Damit stehen sofort alle Features des Spring-Frameworks zur Verfügung
  - Context & Dependency Injection
  - AOP
  - Infrastruktur
    - DataSources
    - Transaktionsmanager

Ein Job in der grafischen Darstellung

- Beispiel: Spring Tools Suite

3.3

JSR 352/JBATCH

Der Job

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="myJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
version="1.0">
    <step id="myStep" >
        <chunk item-count="3">
            <reader ref="myItemReader"/>
            <processor ref="myItemProcessor"/>
            <writer ref="myItemWriter"/>
        </chunk>
    </step>
</job>
```

Fachklassen und Infrastruktur

integrata cegos

- In der Batch-Spezifikation ist die Verwendung eines CDI-Frameworks nicht explizit angegeben
- In der Praxis wird jedoch das Java-CDI-Framework benutzt
  - `@javax.inject.Inject`
  - `@javax.inject.Named`

3.4

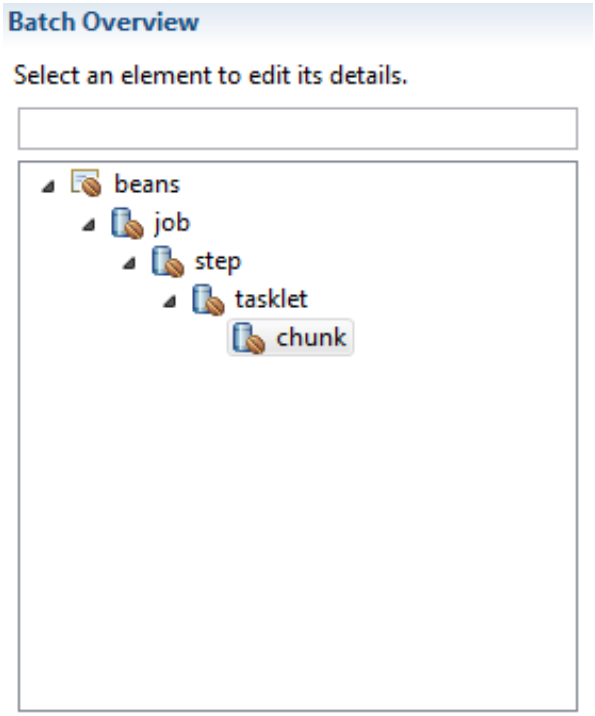
BEFEHLSSATZ

Job-Befehle

- Die Palette des STS zeigt die in Spring Batch verfügbaren Befehle

Step-Struktur

- Ein `Step` selbst besteht
  - aus einem weiteren Sub-Job
  - Einem `Tasklet` (Spring Batch) oder einem `Batchlet` (JSR 352)
    - Diese dienen zur Ausführung einer beliebigen Sequenz von Anweisungen
  - Einem Chunk



Chunks

- Durchlaufen eine festgelegte Sequenz
  - Reader
  - Processor
  - Writer
- Das Einlesen und Schreiben der Daten erfolgt blockweise
  - Bei Bedarf
    - Transaktionell
    - Mit Restart/Retry
- Chunks sind ein großer Unterschied zwischen Batch- und Business-Prozessen

3.5

JAVA API

Chunk-Unterstützung

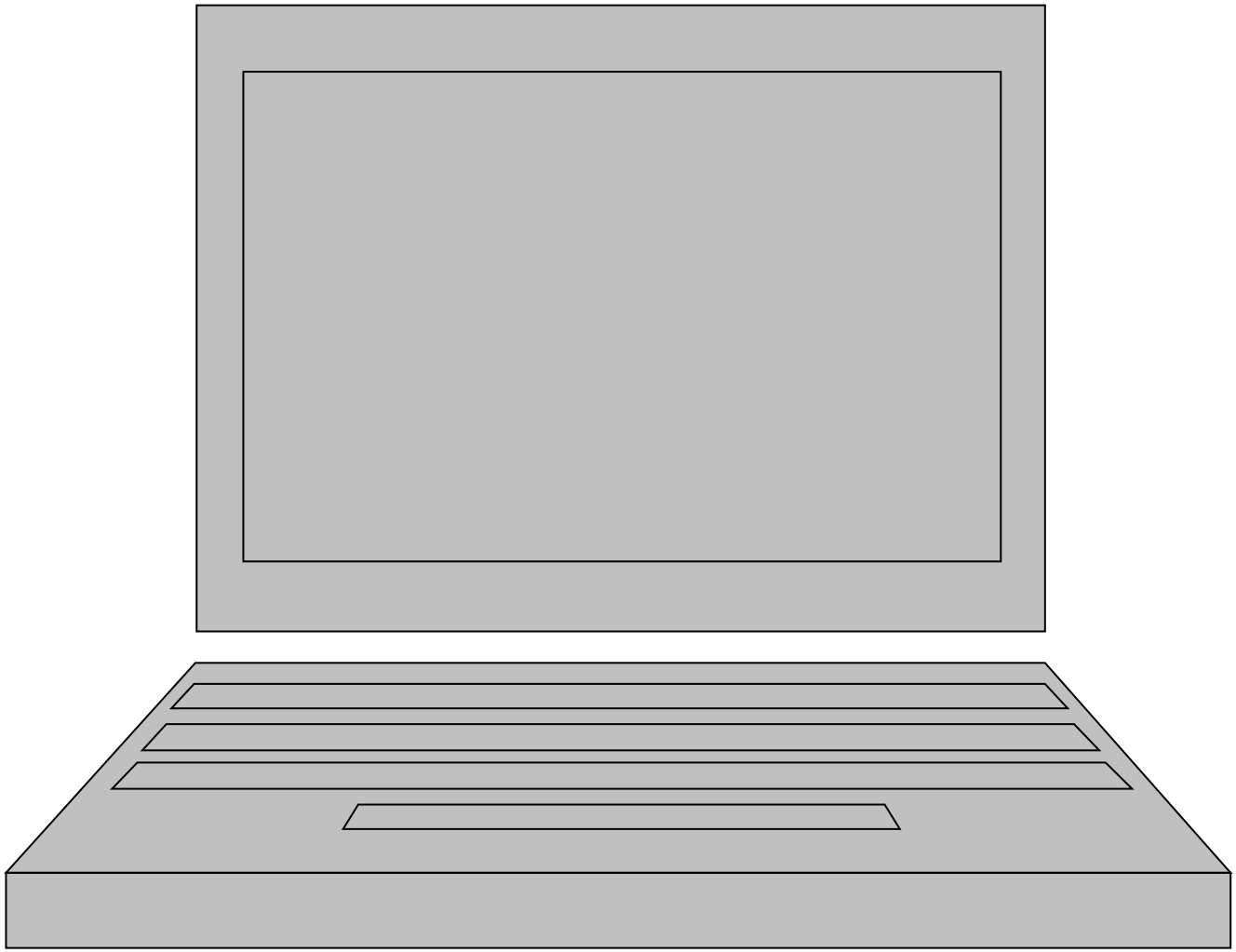
integrata cegos

- Schnittstellen für
  - `ItemReader`
  - `ItemWriter`
  - `ItemProcessor`
- Gleich benannt für Spring Batch und JSR 352
  - Nur unterschiedliche Packages

Listener

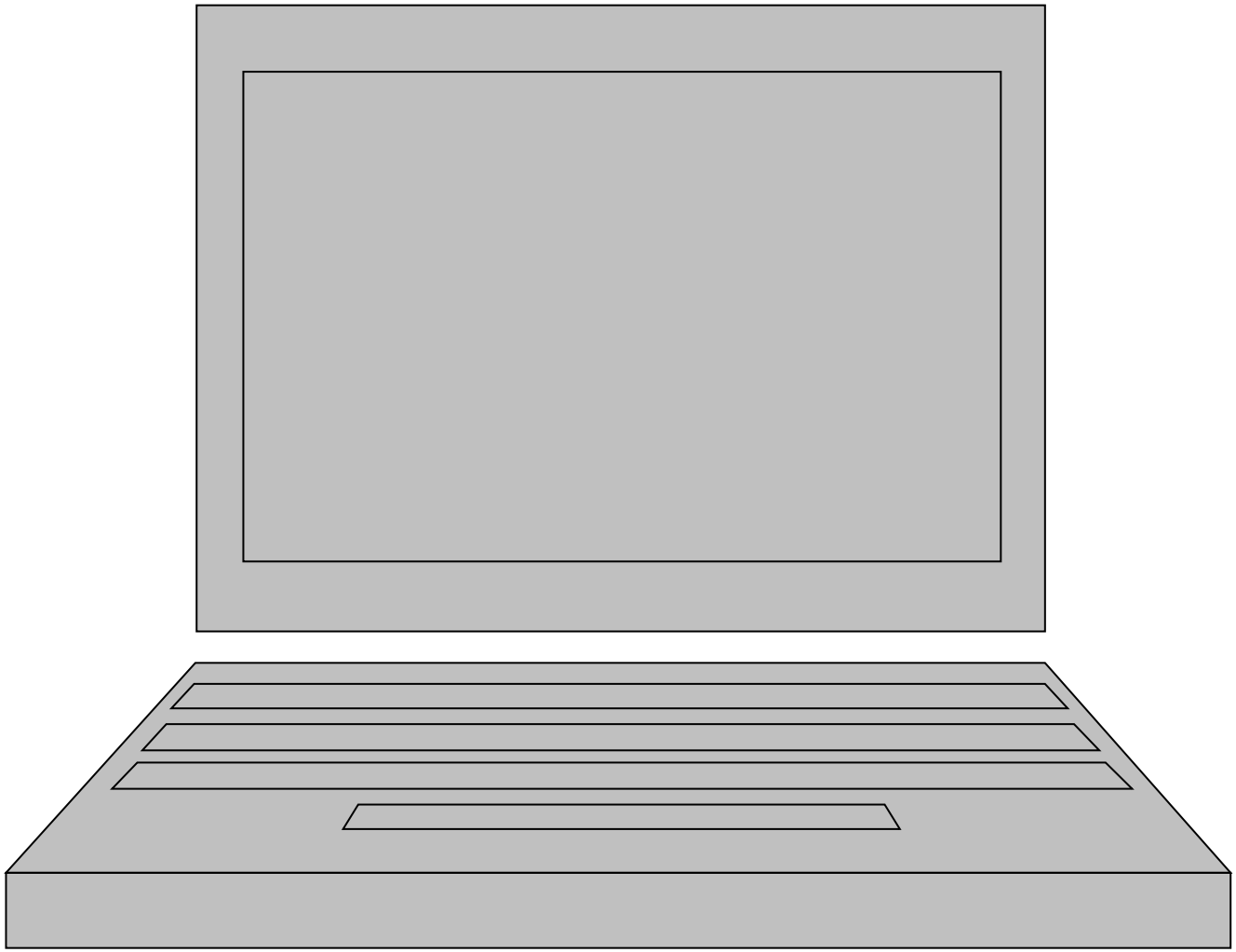
- Der Job-Lauf kann durch Listener detailliert verfolgt werden
  - JobListener
  - StepListener
  - RepeatListener
  - …

Javadoc für Spring Batch



Javadoc für JSR 352

integrata cegos



3.6

TESTEN MIT SPRING BATCH

Testen mit Spring Batch

integrata cegos

```
import org.springframework.batch.test.JobLauncherTestUtils;
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "/simple-job-launcher-
context.xml", "/jobs/simpleJob.xml","/job-runner-
context.xml" })

public class JobStepFunctionalTests {
	@Autowired
	private JobLauncherTestUtils jobLauncherTestUtils;
	@Test
	public void testJobLaunch() throws Exception {
		jobLauncherTestUtils.launchJob();
		//Assertions
	}
```

Testen mit JSR 352

integrata cegos

```
import javax.batch.operations.JobOperator;
public class ChunkSimpleTest {
    @Test
    public void testChunkSimple() throws Exception {
        JobOperator jobOperator =
BatchRuntime.getJobOperator();
       //Assertions
    }
```

3.7

JOB-STATE

Der ExecutionContext

- Persistente Key-Value-Paare
  - eine persistente Map
  - Jede Java-Anwendung als Bestandteil einer Job-Ausführung können Daten im `ExecutionContext` ablegen
    - Und stehen damit beispielsweise nach einem Restart zur Verfügung

3.8

ABLAUFSTEUERUNG

Elementare Job-Befehle

- `decision`
- `flow`
- `split`
- `fail`
- `next`
- `stop`

Retry

- Chunk-Konfiguration

```
<chunk reader="itemGenerator"
        writer="itemWriter"
        commit-interval="1"
       retry-limit="3">
</chunk>
```

Restart

- Job-Konfiguration

```
<job  xmlns=http://www.springframework.org/schema/batch
      id="restartSampleJob"
      restartable="true" >
```

Partitionierung

- Die Eingangsdaten werden durch einen Partitioner in verschiedene disjunkte Partitionen getrennt
  - Strategien
    - Fixed and Even Breakup
    - Aufteilung via Schlüsselspalte
    - Partitionierungstabelle
    - Wertetabelle
    - Aufteilung via Database View
    - Processing Indicator
    - Table to Flat File
    - Hashing Column

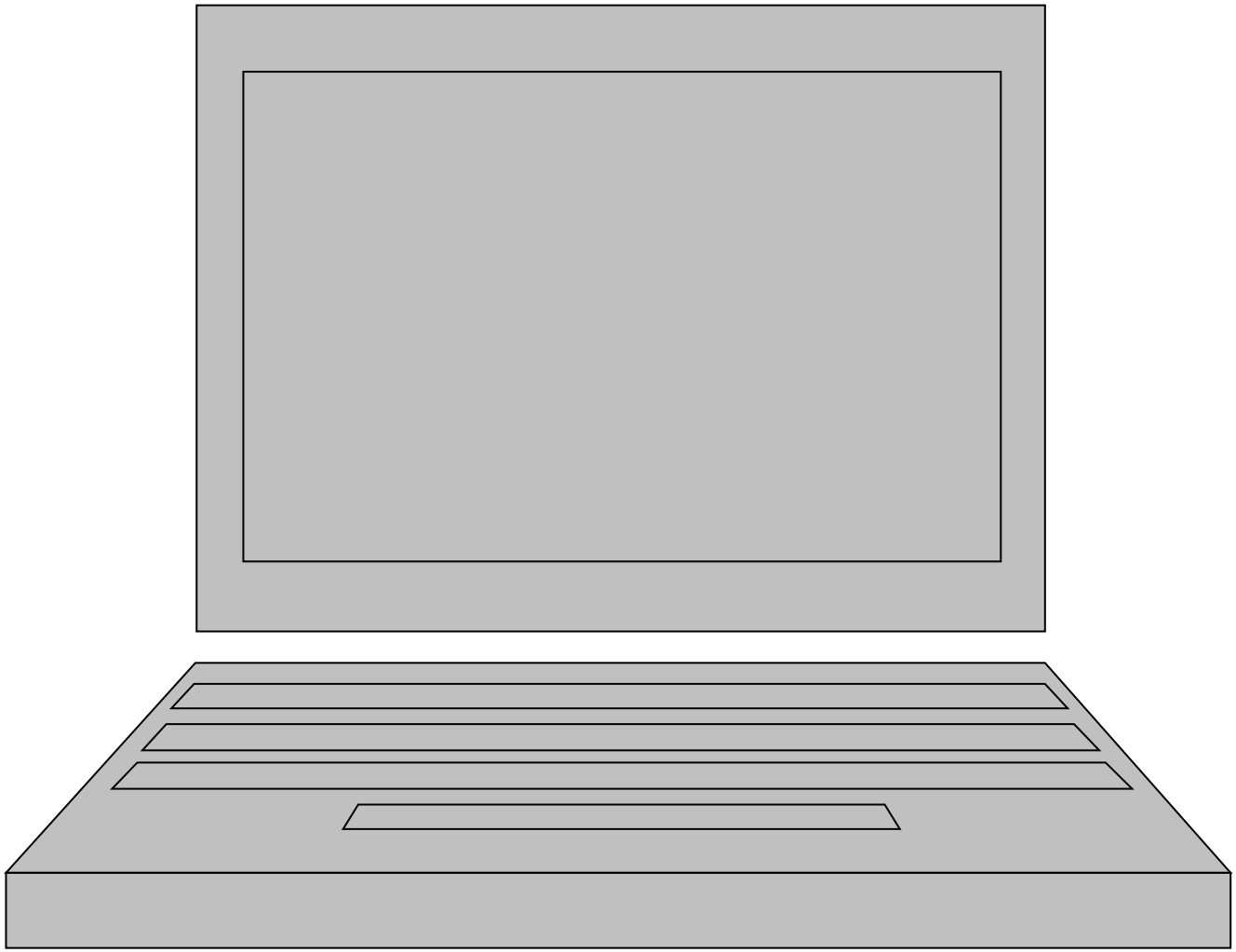
- Ähnlichkeiten zum Map/Reduce-Algorithmus vorhanden
  - Auch die Abfrage einer komplexen NoSQL-Datenbank erfolgt nach Batch-Algorithmen

4

BEISPIELE

4.1

SPRING BATCH



4.2

JSR 352