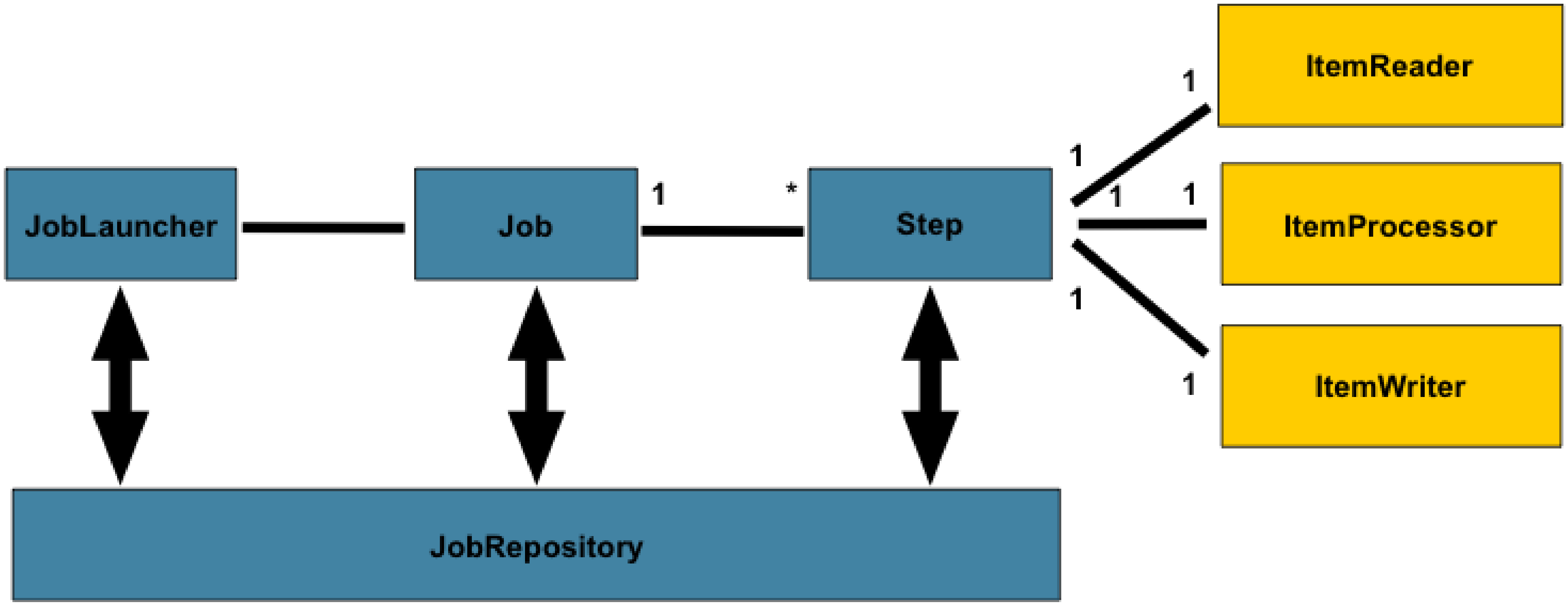
Spring Batch 101

Introduction

- Requirements
  - Automated, complex processing of large volumes of information efficiently processed with out human intervention.
  - Periodic application of complex business rules across large data sets.
  - Integration of information received from internal and external systems that typically requires formatting, validation and processing in a transactional manner into the system.
  - Extraction of large data set for exporting to external systems.


JobLauncher
Job
1
*
Step
1
1
1
1
1
1
ItemReader
ItemProcessor
ItemWriter
JobRepository

Configuring a Job (basic)

```
@Bean
public Job footballJob(JobRepository jobRepository,
   Step playerLoad, Step gameLoad, Step playerSummarization) {
   return new JobBuilder(“footballJob”, jobRepository)
                 .start(playerLoad)
                 .next(gameLoad)
                 .next(playerSummarization)
                 .build();
}
```

Configuring a Job (non restartable)

```
@Bean
public Job footballJob(JobRepository jobRepository,
   Step playerLoad, Step gameLoad, Step playerSummarization) {
   return new JobBuilder(“footballJob”, jobRepository)
                 .preventRestart()
                 .start(playerLoad)
                 .next(gameLoad)
                 .next(playerSummarization)
                 .build();
}
```

Java Configuration

```
@Configuration
@EnableBatchProcessing(
    dataSourceRef = “batchDataSource”,
    transactionManagerRef=”batchTransactionManager”)
public class BatchConfiguration {

  @Bean
  public DataSource batchDataSource() {
     ...
  }

  @Bean
  public TransactionManager batchTransactionManager(DataSource batchDataSource) {
     ...
  }
}
```

Java Configuration

```
@EnableBatchProcessing(
    dataSourceRef = “batchDataSource”,
    transactionManagerRef=”batchTransactionManager”)
```

Defines:

- JobRepository –
- JobLauncher
- JobRegistry
- JobExplorer
- JobOperator

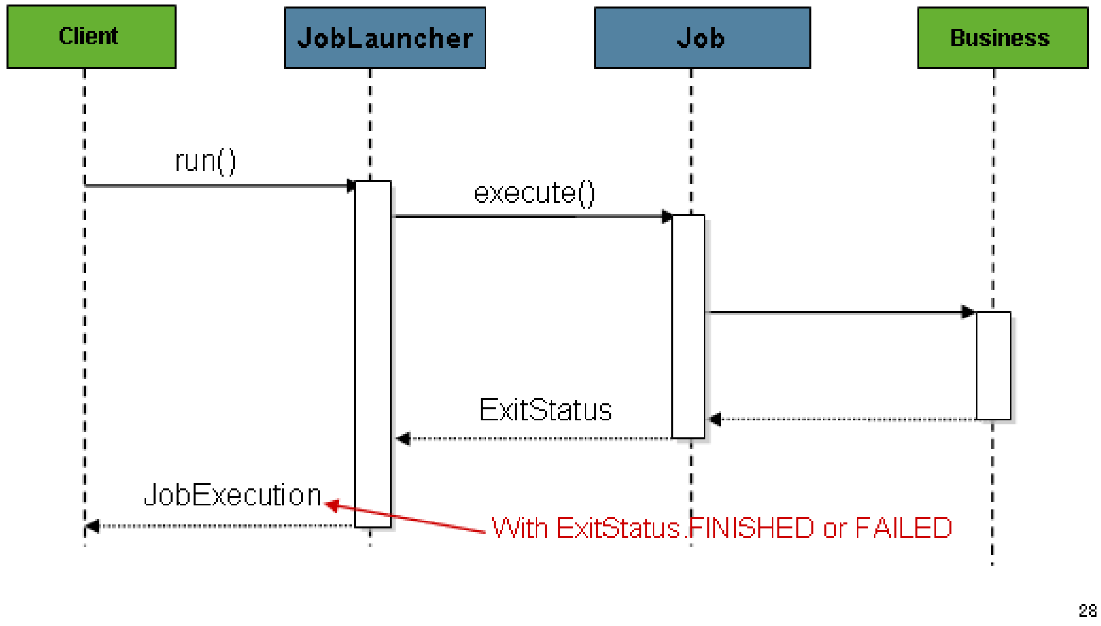
Java Configuration

```
@Configuration
public class BatchConfiguration extends DefaultBatchConfiguration {

   @Bean
   public Job jobName(JobRepository jobRepository, Step start) {
      return new JobBuilder("job-name", jobRepository)
         .preventRestart() // Disables restartability
         // Define job flow as needed
         .build();
   }
}
```

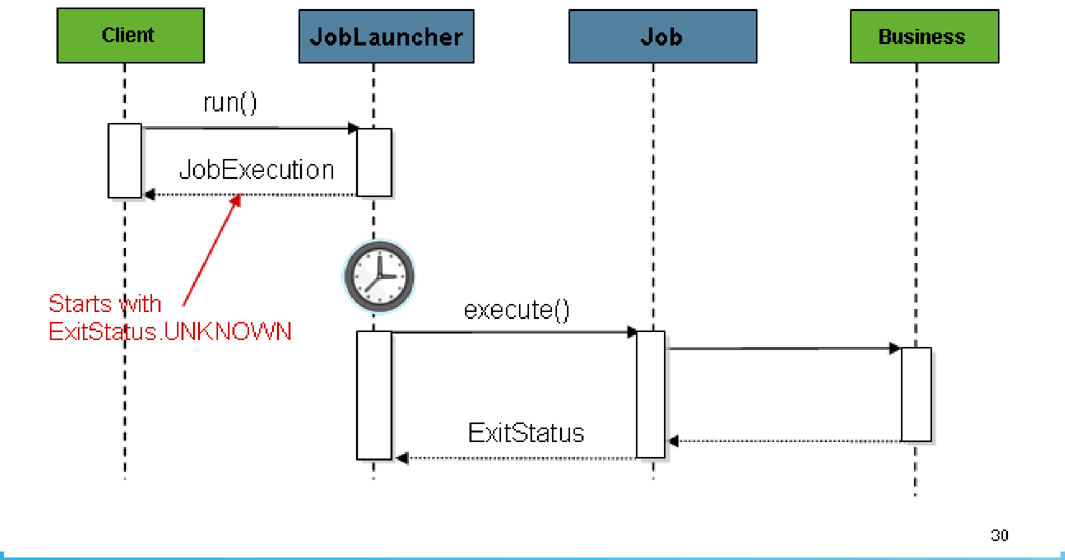
JobLauncher (Sync)

```
@Bean
public JobLauncher jobLauncher(JobRepository jobRepository) throws Exception {
    TaskExecutorJobLauncher jobLauncher = new TaskExecutorJobLauncher();
    jobLauncher.setJobRepository(jobRepository);
    jobLauncher.afterPropertiesSet();
    return jobLauncher;
}
```



JobLauncher (Async)

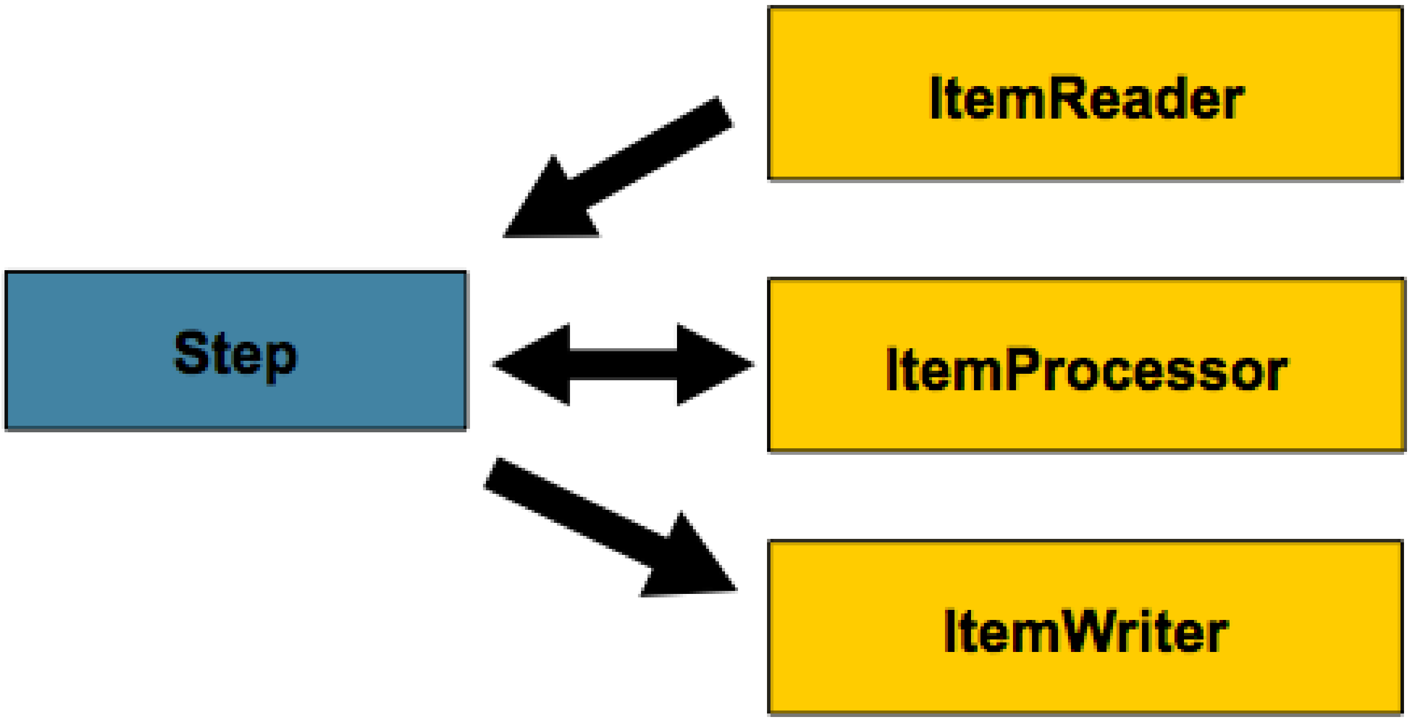
```
@Bean
public JobLauncher jobLauncher(JobRepository jobRepository) throws Exception {
    TaskExecutorJobLauncher jobLauncher = new TaskExecutorJobLauncher();
    jobLauncher.setJobRepository(jobRepository);
    jobLauncher.setTaskExecutor(new SimpleAsyncTaskExecutor());
    jobLauncher.afterPropertiesSet();
    return jobLauncher;
}
```



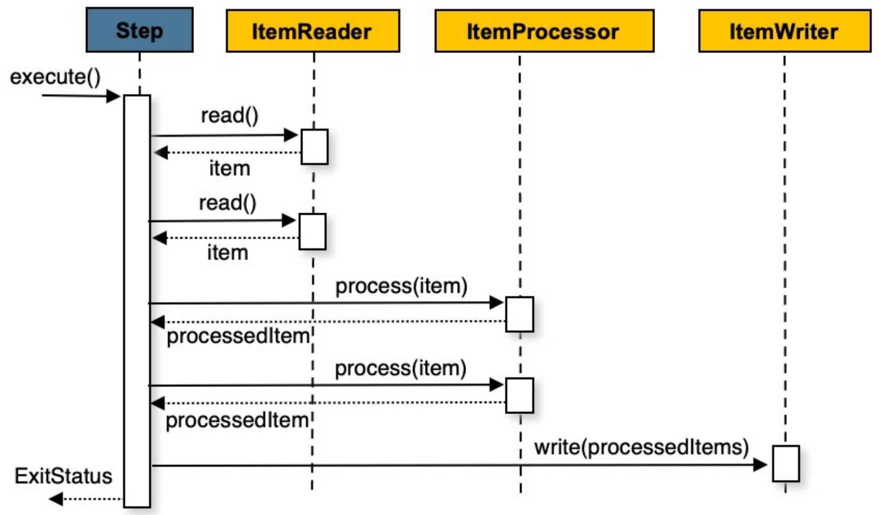
Querying the Repository

```
interface JobExplorer {
    List<JobInstance> getJobInstances(String jobName, int start, int count);
    JobExecution getJobExecution(Long executionId);
    StepExecution getStepExecution(Long jobExecutionId, Long stepExecutionId);
    JobInstance getJobInstance(Long instanceId);
    List<JobExecution> getJobExecutions(JobInstance jobInstance);
    Set<JobExecution> findRunningJobExecutions(String jobName);
}
```

Configuring a Step



Chunk oriented processing



```
var step = new StepBuilder("stepName", jobRepository)
    .<?, ?>chunk(2, transactionManager)
    .reader(reader())
    .itemProcessor(processor())
    .writer(writer())
    .build();
```

ItemReader and ItemWriter

```
public interface ItemReader<T> {
    T read() throws Exception, UnexpectedInputException, ParseException, NonTransientResourceException
}
```

```
public interface ItemWriter<T> {
    void write(Chunk<? extends T> items) throws Exception;
}
```

ItemStream

```
public interface ItemStream {
    void open(ExecutionContext executionContext) throws ItemStreamException;

    void update(ExecutionContext executionContex) thows ItemStreamException;

    void close() throws ItemStreamException;
}
```

Flat Files

```
FieldSet
String[] tokens = new String[]{"foo", "1",
"true"};
FieldSet fs = new DefaultFieldSet(tokens);
String name = fs.readString(0);
int value = fs.readInt(1);
boolean booleanValue = fs.readBoolean(2);
```

```
public interface LineMapper<T> {
    T mapLine(String line, int lineNumber) throws
Exception;
}
```

Flat Files

`FlatFileItemReader`

`FlatFileItemWriter`

More

Creating custom ItemReaders and ItemWriters

Online documentation

Questions

Thanks