Consuming an Input File

Michael Hoffman
DEVELOPER AND PLURALSIGHT AUTHOR

@mhi_inc github.com/michaelhoffmantech

Input File Structure

Comma delimited fields

Header row

File considerations

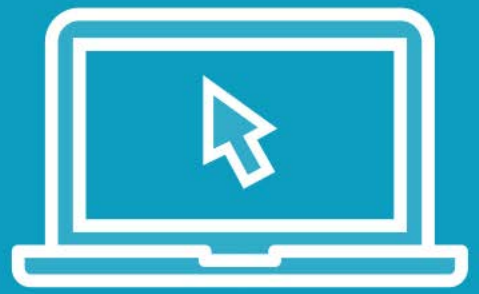
PatientRecord Class

```
public class PatientRecord implements Serializable {


    private String sourceId;
    private String firstName;
    private String middleInitial;
    private String lastName;
    private String emailAddress;
    private String phoneNumber;
    private String street;
    private String city;
    private String state;
    private String zip;
    private String birthDate;
    private String action;
    private String ssn;

}
```

Demo



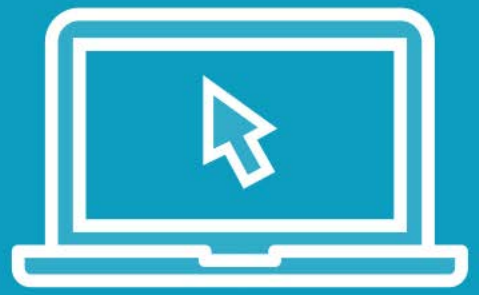
Demo 9 – Input file domain object creation

```
@Bean
public Step step(ItemReader<PatientRecord> itemReader)
    throws Exception {
    return this.stepBuilderFactory
        .get(Constants.STEP_NAME)
        .<PatientRecord, PatientRecord>chunk(2)
        .reader(itemReader)
        .processor(processor())
        .writer(writer())
        .build();
}
```

BatchJobConfiguration.java

Package com.pluralsight.springbatch.patientbatchloader.config

Demo



Demo 10 – Update the job step for chunk-oriented processing

FlatFile ItemReader Builder

Name

Resource

Lines to skip

Line mapper

```
@Bean
@StepScope
public FlatFileItemReader<PatientRecord> reader(
     @Value("#{jobParameters['" + Constants.JOB_PARAM_FILE_NAME +
          "']}")String fileName) {
          return new FlatFileItemReaderBuilder<PatientRecord>()
               .name(Constants.ITEM_READER_NAME)
               .resource(new PathResource(Paths.get(
                    applicationProperties.getBatch().getInputPath() +
                         File.separator + fileName)))
               .linesToSkip(1)
               .lineMapper(lineMapper())
               .build();
}
```

BatchJobConfiguration.java

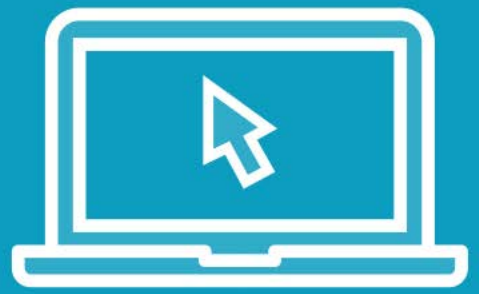
Package com.pluralsight.springbatch.patientbatchloader.config

```
@Bean
public LineMapper<PatientRecord> lineMapper() {
     DefaultLineMapper<PatientRecord> mapper = new DefaultLineMapper<>();
     mapper.setFieldSetMapper((fieldSet) -> new PatientRecord(
          fieldSet.readString(0), fieldSet.readString(1),
          fieldSet.readString(2), fieldSet.readString(3),
          fieldSet.readString(4), fieldSet.readString(5),
          fieldSet.readString(6), fieldSet.readString(7),
          fieldSet.readString(8), fieldSet.readString(9),
          fieldSet.readString(10), fieldSet.readString(11),
          fieldSet.readString(12)));
     mapper.setLineTokenizer(new DelimitedLineTokenizer());
     return mapper;
}
```

BatchJobConfiguration.java

Package com.pluralsight.springbatch.patientbatchloader.config

Demo



Demo 11 – Implementing a flat file item reader

```
@Bean
@StepScope
public PassThroughItemProcessor<PatientRecord> processor() {

    return new PassThroughItemProcessor<>();

}
```

BatchJobConfiguration.java

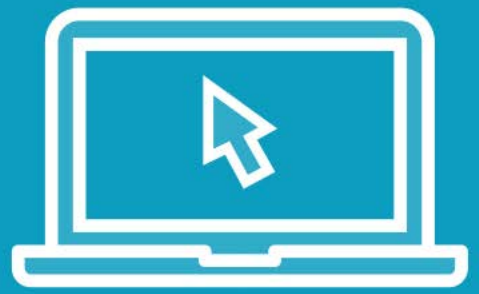
Package com.pluralsight.springbatch.patientbatchloader.config

```
@Bean
@StepScope
public ItemWriter<PatientRecord> writer() {
     return new ItemWriter<PatientRecord>() {
          @Override
          public void write(List<? extends PatientRecord> items) throws Exception {
               for (PatientRecord patientRecord : items) {
                    System.err.println("Writing item: " + patientRecord.toString());
               }
          }
     };
}
```

BatchJobConfiguration.java

Package com.pluralsight.springbatch.patientbatchloader.config

Demo



Demo 12 – Implementing the stub processor and writer

```
@Autowired
private FlatFileItemReader<PatientRecord> reader;

private JobParameters jobParameters;

@Before
public void setUp() {
     Map<String, JobParameter> params = new HashMap<>();
     params.put(Constants.JOB_PARAM_FILE_NAME,
          new JobParameter("test-unit-testing.csv"));
     jobParameters = new JobParameters(params);
}
```

BatchJobConfigurationTest.java

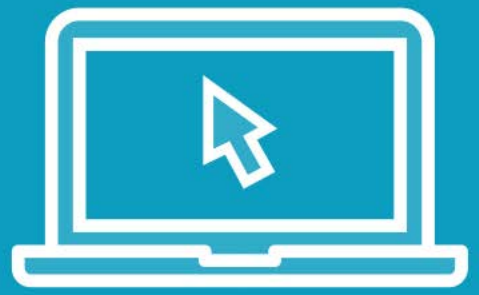
Class in the test folder under the package

com.pluralsight.springbatch.patientbatchloader.config

BatchJobConfigurationTest

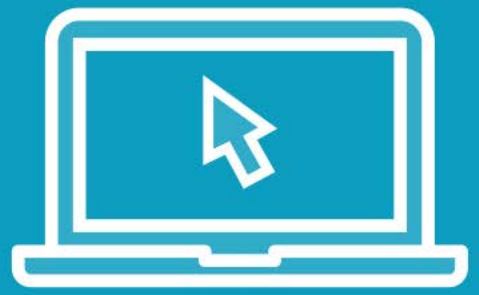
```
@Test
public void testReader() throws Exception {
	StepExecution stepExecution = MetaDataInstanceFactory.createStepExecution(jobParameters);
	int count = 0;
	try {
		count = StepScopeTestUtils.doInStepScope(stepExecution, () -> {
			int numPatients = 0;
			PatientRecord patient;
			try {
				reader.open(stepExecution.getExecutionContext());
				while ((patient = reader.read()) != null) {
					// ASSERTIONS
				}
			} finally {
				try { reader.close(); } catch (Exception e) { fail(e.toString()); }
			}
			return numPatients;
		});
	} catch (Exception e) {
		fail(e.toString());
	}
	assertEquals(1, count);
}
```

Demo



Demo 13 – Testing the item reader

Demo



Demo 14 – Executing the job with the item reader

Summary

- Updated the step for chunking
- Implemented a FlatFileItemReader
- Stubbed the item processor and writer
- Tested the item reader