# Response:

Certainly! Here's a step-by-step translation of the JCL job functionalities, COBOL program logic, subroutine workflows, and Copybooks content into a Spring Batch application. This includes the necessary Java classes, configuration files, and unit tests.

## 1. Spring Batch Configuration

## `Batchconfig.Java`

```java

@Configuration

@EnableBatchProcessing

public class BatchConfig {

@Autowired

private JobBuilderFactory jobBuilderFactory;

@Autowired

private StepBuilderFactory stepBuilderFactory;

@Autowired

private DataSource dataSource;

@Bean

public Job edi850Job(JobCompletionNotificationListener listener, Step step1) {

return jobBuilderFactory.get("edi850Job")

.listener(listener)

.start(step1)

.build();

}

@Bean

public Step step1(ItemReader<InputRecord> reader, ItemProcessor<InputRecord, OutputRecord> processor, ItemWriter<OutputRecord> writer) {

return stepBuilderFactory.get("step1")

.<InputRecord, OutputRecord> chunk(10)

.reader(reader)

.processor(processor)

.writer(writer)

.build();

}

}

```

## `Jobcompletionnotificationlistener.Java`

```java

@Component

public class JobCompletionNotificationListener extends JobExecutionListenerSupport {

private static final Logger log = LoggerFactory.getLogger(JobCompletionNotificationListener.class);

@Override

public void afterJob(JobExecution jobExecution) {

if (jobExecution.getStatus() == BatchStatus.COMPLETED) {

log.info("Job completed successfully");

}

}

}

```

## 2. Java Equivalents Of Cobol Program And Subroutine Logic

## `Inputrecord.Java`

```java

public class InputRecord {

private String recordData;

// Getters and Setters

}

```

## `Outputrecord.Java`

```java

public class OutputRecord {

private String enrichedData;

// Getters and Setters

}

```

## `Inputfilereader.Java`

```java

public class InputFileReader implements ItemReader<InputRecord> {

private BufferedReader reader;

public InputFileReader(String inputFile) {

try {

reader = new BufferedReader(new FileReader(inputFile));

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

@Override

public InputRecord read() throws Exception {

String line = reader.readLine();

if (line == null) {

return null;

}

InputRecord record = new InputRecord();

record.setRecordData(line);

return record;

}

}

```

## `Recordprocessor.Java`

```java

public class RecordProcessor implements ItemProcessor<InputRecord, OutputRecord> {

@Autowired

private JdbcTemplate jdbcTemplate;

@Override

public OutputRecord process(InputRecord inputRecord) throws Exception {

String recordData = inputRecord.getRecordData();

// Implement enrichment logic

String enrichedData = enrichRecord(recordData);

OutputRecord outputRecord = new OutputRecord();

outputRecord.setEnrichedData(enrichedData);

return outputRecord;

}

private String enrichRecord(String recordData) {

// Sample SQL Query to enrich data

String sql = "SELECT additional\_data FROM enrichment\_table WHERE record\_key = ?";

String enrichedData = jdbcTemplate.queryForObject(sql, new Object[]{recordData}, String.class);

return enrichedData;

}

}

```

## `Outputfilewriter.Java`

```java

public class OutputFileWriter implements ItemWriter<OutputRecord> {

private BufferedWriter writer;

public OutputFileWriter(String outputFile) {

try {

writer = new BufferedWriter(new FileWriter(outputFile));

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void write(List<? extends OutputRecord> items) throws Exception {

for (OutputRecord item : items) {

writer.write(item.getEnrichedData());

writer.newLine();

}

writer.flush();

}

}

```

## 3. Mapping Subroutine Workflows To Spring Batch Components

The subroutine for enrichment has been incorporated into the `RecordProcessor` class, where database queries are performed to enrich input records.

## 4. Conversion Of Copybooks Content And Structure

Assuming the Copybooks define the structure of input and output records, we have already defined `InputRecord` and `OutputRecord` classes that hold these structures.

## Unit Tests

## `Batchconfigtest.Java`

```java

@RunWith(SpringRunner.class)

@SpringBootTest

public class BatchConfigTest {

@Autowired

private JobLauncherTestUtils jobLauncherTestUtils;

@Autowired

private JobRepositoryTestUtils jobRepositoryTestUtils;

@Before

public void setUp() throws Exception {

jobRepositoryTestUtils.removeJobExecutions();

}

@Test

public void testJob() throws Exception {

JobExecution jobExecution = jobLauncherTestUtils.launchJob();

assertEquals(BatchStatus.COMPLETED, jobExecution.getStatus());

}

}

```

## `Recordprocessortest.Java`

```java

@RunWith(SpringRunner.class)

@SpringBootTest

public class RecordProcessorTest {

@Autowired

private RecordProcessor processor;

@MockBean

private JdbcTemplate jdbcTemplate;

@Test

public void testProcess() throws Exception {

InputRecord inputRecord = new InputRecord();

inputRecord.setRecordData("some data");

when(jdbcTemplate.queryForObject(anyString(), any(Object[].class), eq(String.class)))

.thenReturn("enriched data");

OutputRecord outputRecord = processor.process(inputRecord);

assertEquals("enriched data", outputRecord.getEnrichedData());

}

}

```

## Summary

This Spring Batch application reads an input file, processes each record to enrich it using data from a database, and writes the enriched records to an output file. The JCL job functionalities are translated into Spring Batch configurations, and the COBOL program logic is implemented in Java classes. Subroutine workflows are integrated into the Spring Batch processor, and Copybooks content is mapped to Java data structures. Unit tests ensure the correctness of the batch job and its components.