### **Key Components**

1. **FlatFileItemWriter** with a custom **LineAggregator**.
2. **Custom Formatter** to handle EBCDIC encoding and packed decimal conversion.
3. **Spring Batch Configuration**.

### **Steps in the Code**

1. **Read data from a database using JdbcCursorItemReader.**
2. **Process data to convert numeric values into packed decimal format.**
3. **Write output to a flat file using a custom writer.**

Now, let's look at the full implementation:

Spring Batch Ebcdic

import org.springframework.batch.core.Job;

import org.springframework.batch.core.Step;

import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;

import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;

import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;

import org.springframework.batch.item.ItemProcessor;

import org.springframework.batch.item.database.JdbcCursorItemReader;

import org.springframework.batch.item.file.FlatFileItemWriter;

import org.springframework.batch.item.file.transform.LineAggregator;

import org.springframework.batch.item.support.ListItemWriter;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.io.FileSystemResource;

import javax.sql.DataSource;

import java.nio.charset.Charset;

import java.nio.charset.StandardCharsets;

import java.text.DecimalFormat;

import java.util.Arrays;

@Configuration

@EnableBatchProcessing

public class EbcdicBatchConfig {

@Bean

public JdbcCursorItemReader<Customer> reader(DataSource dataSource) {

JdbcCursorItemReader<Customer> reader = new JdbcCursorItemReader<>();

reader.setDataSource(dataSource);

reader.setSql("SELECT id, name, balance FROM customers");

reader.setRowMapper((rs, rowNum) -> new Customer(rs.getInt("id"), rs.getString("name"), rs.getDouble("balance")));

return reader;

}

@Bean

public ItemProcessor<Customer, String> processor() {

return customer -> {

// Convert numeric fields to packed decimal

String packedBalance = convertToPackedDecimal(customer.getBalance(), 10, 2);

// Encode name field in EBCDIC

byte[] ebcdicName = customer.getName().getBytes(Charset.forName("Cp500"));

return String.format("%-10s%s", new String(ebcdicName, Charset.forName("Cp500")), packedBalance);

};

}

@Bean

public FlatFileItemWriter<String> writer() {

FlatFileItemWriter<String> writer = new FlatFileItemWriter<>();

writer.setResource(new FileSystemResource("output.ebc"));

writer.setLineAggregator(new LineAggregator<String>() {

@Override

public String aggregate(String item) {

return item;

}

});

writer.setEncoding("Cp500"); // EBCDIC Encoding

return writer;

}

@Bean

public Step step(StepBuilderFactory stepBuilderFactory, JdbcCursorItemReader<Customer> reader,

ItemProcessor<Customer, String> processor, FlatFileItemWriter<String> writer) {

return stepBuilderFactory.get("step")

.<Customer, String>chunk(10)

.reader(reader)

.processor(processor)

.writer(writer)

.build();

}

@Bean

public Job job(JobBuilderFactory jobBuilderFactory, Step step) {

return jobBuilderFactory.get("ebcdicJob")

.start(step)

.build();

}

// Helper method to convert double values to packed decimal format

private String convertToPackedDecimal(double value, int length, int scale) {

DecimalFormat df = new DecimalFormat("#" + (scale > 0 ? "." + "0".repeat(scale) : ""));

String formattedValue = df.format(value).replace(".", "");

String hexPackedDecimal = formattedValue + "C"; // 'C' indicates positive number in EBCDIC

StringBuilder packed = new StringBuilder();

for (int i = 0; i < hexPackedDecimal.length(); i += 2) {

packed.append((char) Integer.parseInt(hexPackedDecimal.substring(i, i + 2), 16));

}

return packed.toString();

}

}

99

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

import org.springframework.batch.core.Job;

import org.springframework.batch.core.Step;

import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;

import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;

import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;

import org.springframework.batch.item.ItemProcessor;

import org.springframework.batch.item.database.JdbcCursorItemReader;

import org.springframework.batch.item.file.FlatFileItemWriter;

import org.springframework.batch.item.file.transform.LineAggregator;

import org.springframework.batch.item.support.ListItemWriter;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.io.FileSystemResource;

import javax.sql.DataSource;

import java.nio.charset.Charset;

import java.nio.charset.StandardCharsets;

import java.text.DecimalFormat;

import java.util.Arrays;

@Configuration

@EnableBatchProcessing

public class EbcdicBatchConfig {

@Bean

public JdbcCursorItemReader<Customer> reader(DataSource dataSource) {

JdbcCursorItemReader<Customer> reader = new JdbcCursorItemReader<>();

reader.setDataSource(dataSource);

reader.setSql("SELECT id, name, balance FROM customers");

reader.setRowMapper((rs, rowNum) -> new Customer(rs.getInt("id"), rs.getString("name"), rs.getDouble("balance")));

return reader;

}

@Bean

public ItemProcessor<Customer, String> processor() {

return customer -> {

Answer in chat instead

### **Explanation**

1. **JdbcCursorItemReader**: Reads id, name, and balance from a database.
2. **ItemProcessor**:  
   * Converts balance into **packed decimal format**.
   * Converts name into **EBCDIC encoding** (Cp500 code page).
3. **FlatFileItemWriter**:  
   * Writes data to an EBCDIC encoded file (Cp500).
   * Uses LineAggregator to properly format lines.

### **Packed Decimal Example**

A decimal value like **123.45** in packed decimal format (10,2) will be stored as:

r

CopyEdit

0x12 0x34 0x5C (C = positive sign in EBCDIC)

This ensures correct processing of **mainframe-compatible flat files**.

Would you like to add more fields or change the format?