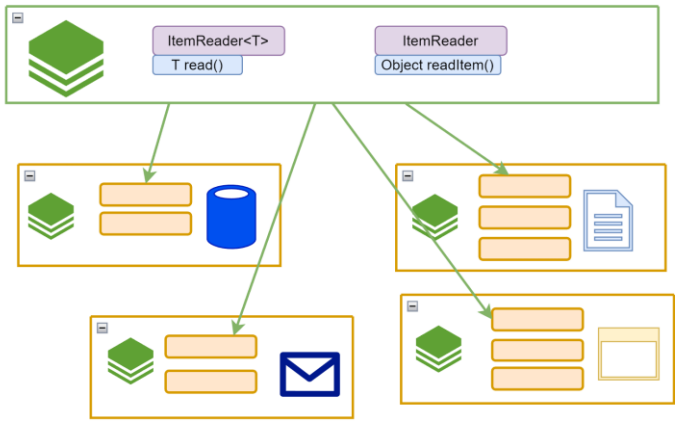
Leitores do Spring Batch

Todos os **leitores** do **Spring Batch** herdam da **Interface ItemReader<T>** *(que pode ou não ser parametrizada),* essa interface possuí um método principal chamado ***read()*** que é invocado enquanto retornar dados ou atingir o tamanho do chunck (o size predefinido dele) ou retornar nullo.

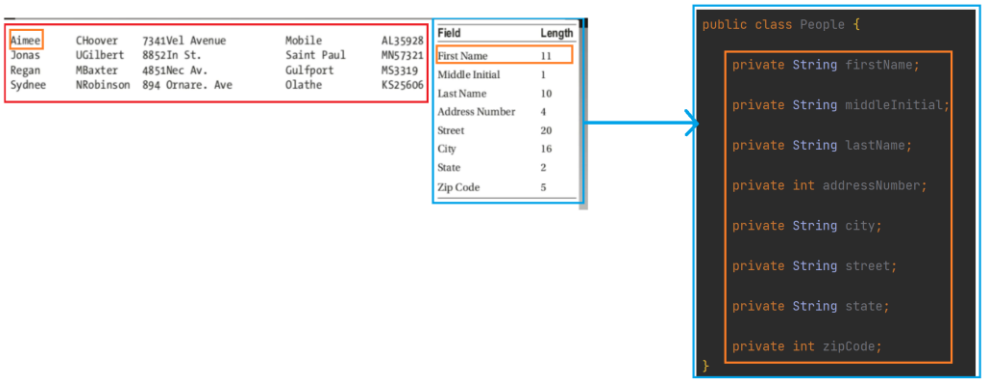
A implementação do **leitor** depende de que **fonte de dados** a gente quer ler, um *XML, um txt um webRequest, banco de dados etc....* O Spring Batch possuí **implementações prontas** para a maioria absoluta dos casos:



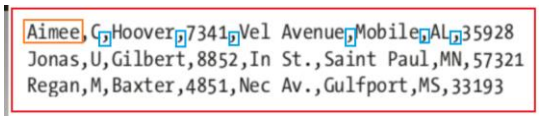
Leitura de arquivos flat

Acho que é interessante entender o que é um **arquivo flat**, um arquivo flat é um arquivo que possuí ***dados não estruturados***. E ***"dados não estruturados"*** simplesmente quer dizer que quando olhamos o conteúdo não dá pra saber o formato ou significado do que está representado **ali**.

Um exemplo de **arquivo flat** são os baseados em **largura fixa**, ou seja, cada **propriedade** do arquivo possuí um número delimitado de colunas ***(por exemplo, um nome com 11 caracteres vai ocupar 11 colunas no arquivo)***, e todas essas colunas são incluídas no *mapeamento para* **o objeto de domínio:**



Outro exemplo de arquivo flat são os **delimitados**, basicamente a cada **delimitador** eu tenho uma **propriedade**:



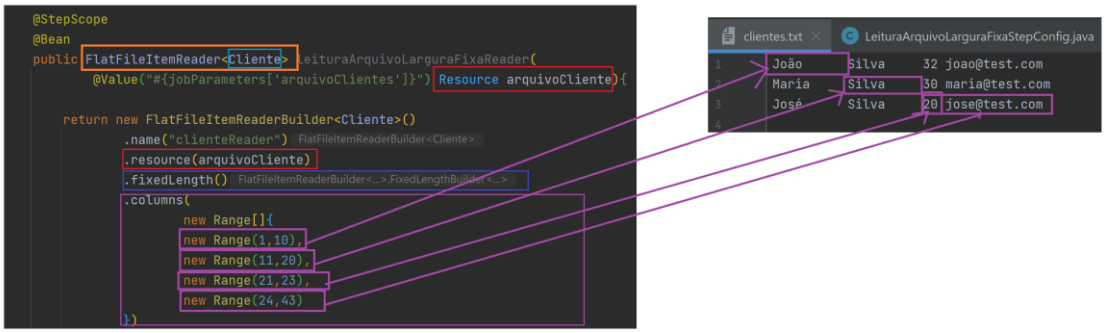
E tem um tipo de **arquivo flat** ainda mais complexo, o que envolve ***múltiplos formatos***, e nesse caso eu to falando de um arquivo que vai conter **mais de um objeto de domínio**. Por exemplo esse que tem tanto o domínio de **Customer** quanto de **Transaction**:

```
CUST,Warren,Q,Darrow,8272 4th Street,New York,IL,76091
TRANS,1165965,2011-01-22 00:13:29,51.43
CUST,Ann,V,Gates,9247 Infinite Loop Drive,Hollywood,NE,37612
CUST,Erica,I,Jobs,8875 Farnam Street,Aurora,IL,36314
TRANS,8116369,2011-01-21 20:40:52,-14.83
TRANS,8116369,2011-01-21 15:50:17,-45.45
TRANS,8116369,2011-01-21 16:52:46,-74.6
TRANS,8116369,2011-01-22 13:51:05,48.55
TRANS,8116369,2011-01-21 16:51:59,98.53
```

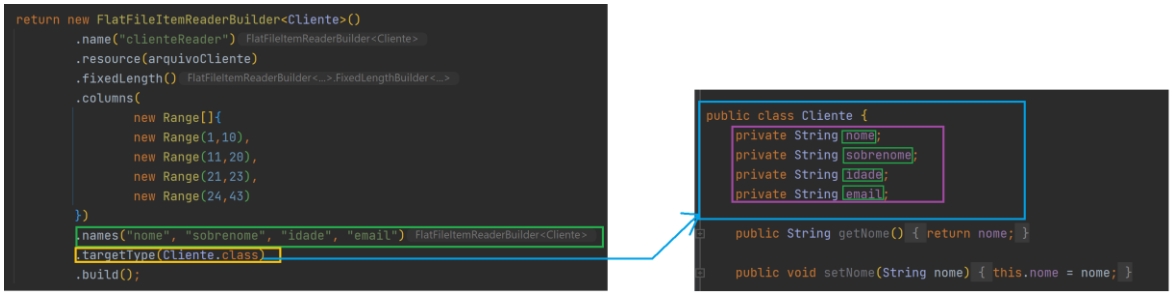
Lendo um arquivo flat de tamanho fixo

Já que o **Spring Batch** tem uma implementação para leituras de arquivo flat, vamos usa-la **(FlatFileItemReader<T>)** para ler um arquivo do tipo **"largura fixa"** (aquele de colunas).

A configuração desse leitor pode ser feita através de um builder, onde eu digo o **resource(nesse caso o arquivo)** fonte da leitura, dizemos que **ele é do tipo "tamanho fixo"**, aí **mapeamos o range das colunas (de que coluna até que coluna representa um atributo, e o próximo, e o próximo) com um array de ranges**:



Em seguida **nomeamos esses ranges com um array de strings (aqui é um objeto de domínio, então obviamente precisam ser os mesmos nomes que o objeto de domínio tem nos atributos)**, com o builder já podemos então informar pra **qual classe devemos converter esse arquivo e pronto:**



Em uma visão geral temos a seguinte configuração: um **package para o job**, **para o step**, **para o reader e para o writer**. Mantendo a estrutura padrão, um job tem Steps, steps podem ser chuncks que por sua vez trabalham com leitura, processamento e escrita. O Chunk aqui está definido como 1 por vez:

Outro ponto interessante aqui é a anotação **@StepScope** no reader, isso acontece porque estou **injetando um valor nesse Reader através de um argumento**, mas para que esse valor seja lido eu preciso dizer que o Reder está no escopo do Step, já que o batch chama o job que chama o step:

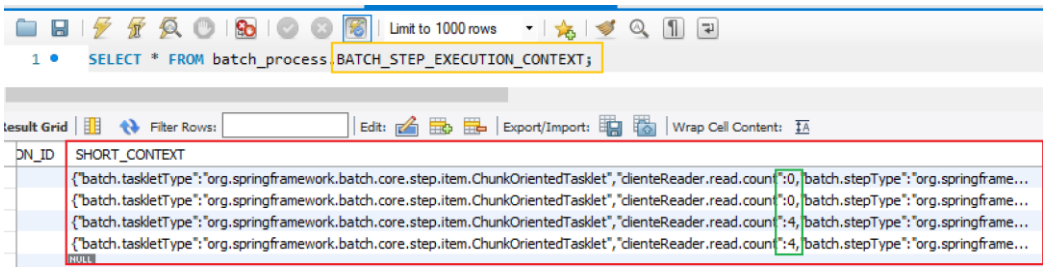
Restart na leitura

O ponto primordial dos **restarts do job em caso de falha** é o tamanho do chunk. O tamanho do chunk(pedaço) **controla a transação**, se meu **chunk** tem tamanho 1, minhas transações/commits vão ser feitos de 1 em 1, se o chunk tem tamanho 10 as transações/commits vão ser feitas de 10 em 10.

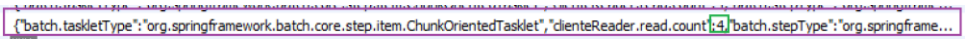
Esse balanço de tamanho do **chunk** é interessante pra questões de performances, mas também influencia diretamente em como o **job** vai ser reexecutado em caso de falha, o processamento considera o chunk como uma unidade de contagem, ou seja, a cada **chunk completo** a informação é salva nos metadados, permitindo ao **Spring Batch** recomeçar o **job** a partir do **chunk subsequente ao último processado com sucesso**.

Por exemplo, se eu tenho **100 registros** divididos em **chunks de 10**, eu vou ter ao todo **10 chunks** com **10 registros** para processar, se meus **4 primeiros chunks** funcionam corretamente, mas o **5 chunk** falha no processamento do seu **9 registro** então o **chunk todo é considerado como falha** e o **Spring Batch** vai saber que 4 chunks deram certo e vai começar a partir do 5 em diante.

O que representa a quantidade de chunks executados no banco de dados é o atributo **SHORT_CONTEXT** da tabela **BATCH_STEP_EXECUTION_CONTEXT**, nele podemos ver o **count** que representa qual o próximo chunk daquele step a ser executado:



nesse **exemplo aqui** eu li um arquivo com 3 registros onde cada **chunk tinha tamanho igual a 1**, depois de executar os **3 chunks com sucesso**, o próximo **chunk é o 4:**



Aqui **nenhum chunk foi feito com sucesso**, o que indica que o **próximo chunk desse step é o 0:**