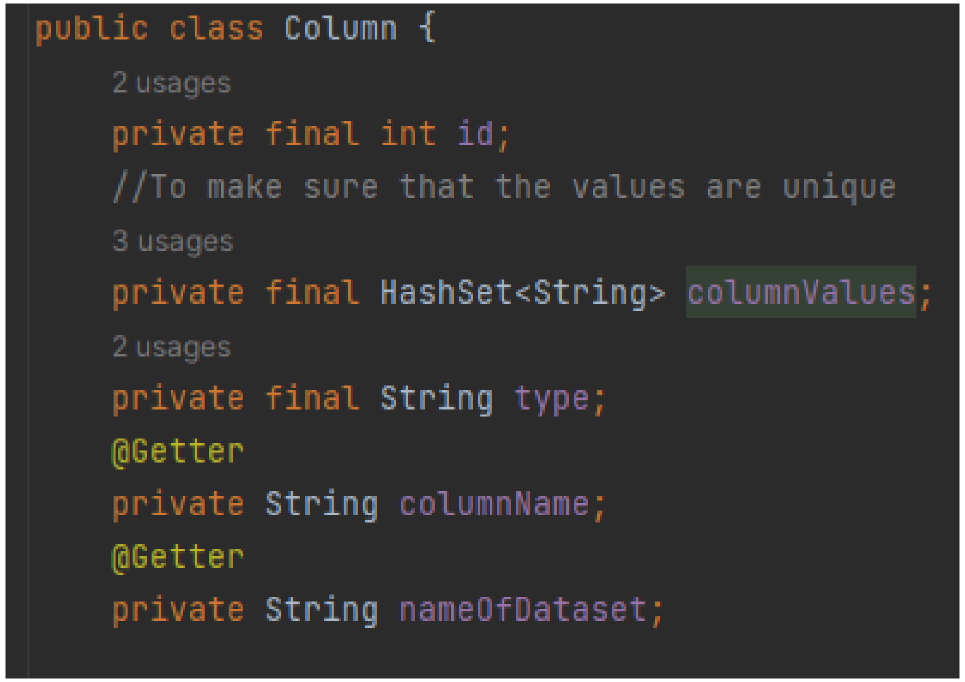
Akka

Tobias Chisi, Nureldien Gebril

Our Idea

- Create a column class
- Create two buckets string and number for column type
- Created composite key class with subkeys Dataset name and Column name
- Search though each bucket to find dependency

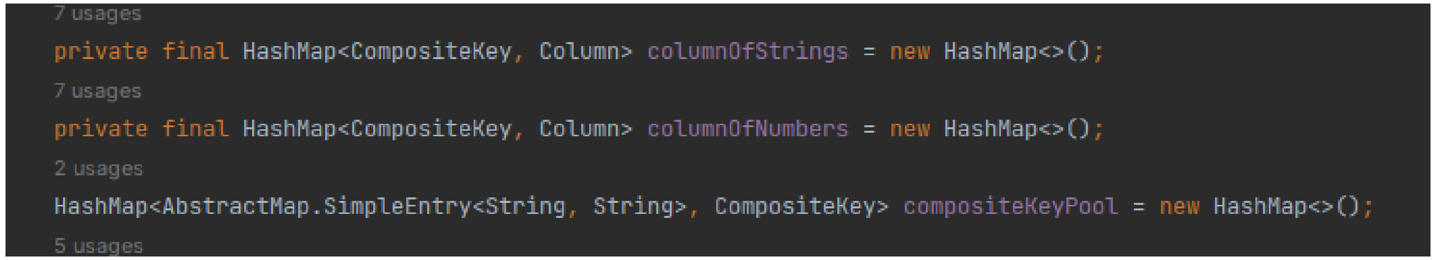


```
public class Column {
    2 usages
    private final int id;
    //To make sure that the values are unique
    3 usages
    private final HashSet<String> columnValues;
    2 usages
    private final String type;
    @Getter
    private String columnName;
    @Getter
    private String nameOfDataset;
```

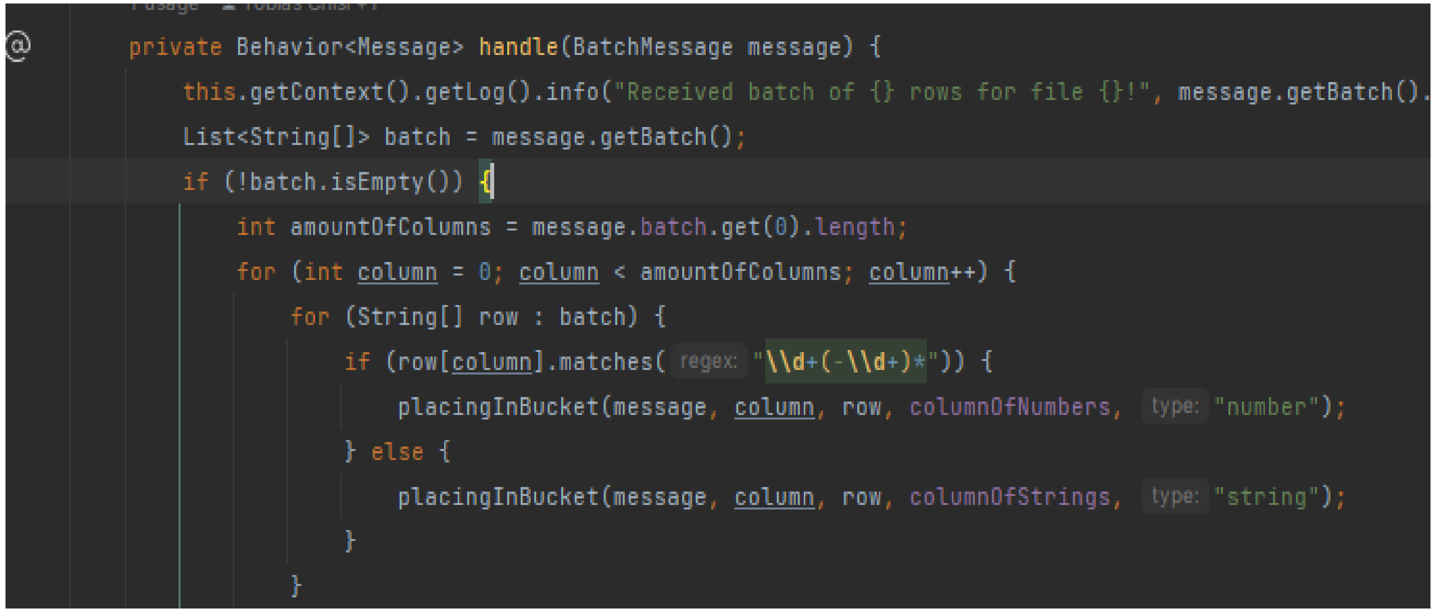
Column Class

- Column class has 5 fields
- columnValues for column values (No duplicates)
- Type (there are two types, numbers and String)
- columnName is the headerline
- columnDataset (from inputFiles[message.getId()])

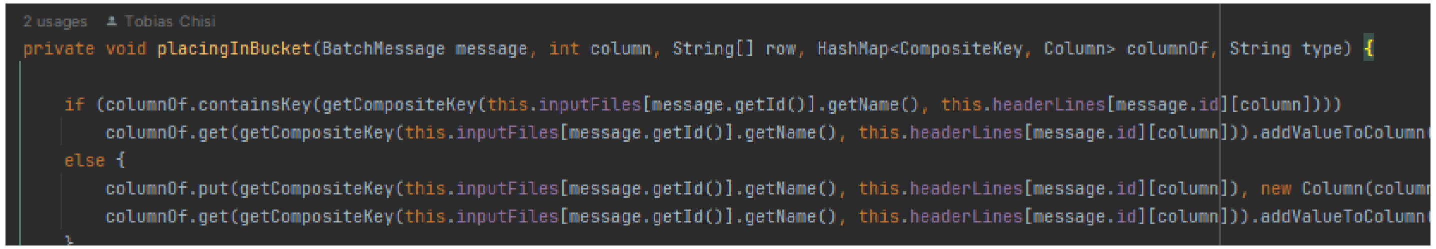
Filling buckets



```
7 usages
private final HashMap<CompositeKey, Column> columnOfStrings = new HashMap<>();
7 usages
private final HashMap<CompositeKey, Column> columnOfNumbers = new HashMap<>();
2 usages
HashMap<AbstractMap.SimpleEntry<String, String>, CompositeKey> compositeKeyPool = new HashMap<>();
5 usages
```



```
    private Behavior<Message> handle(BatchMessage message) {
        this.getContext().getLog().info("Received batch of {} rows for file {}!", message.getBatch().
        List<String[]> batch = message.getBatch();
        if (!batch.isEmpty()) {
            int amountOfColumns = message.batch.get(0).length;
            for (int column = 0; column < amountOfColumns; column++) {
                for (String[] row : batch) {
                    if (row[column].matches( regex: "\\d+(-\\d+)*")) {
                        placingInBucket(message, column, row, columnOfNumbers,  type: "number");
                    } else {
                        placingInBucket(message, column, row, columnOfStrings,  type: "string");
                    }
                }
```

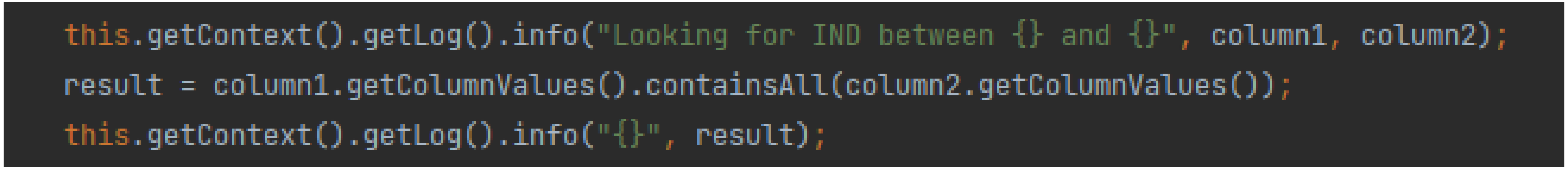


```
2 usages  Tobias Chisi
private void placingInBucket(BatchMessage message, int column, String[] row, HashMap<CompositeKey, Column> columnOf, String type) {

    if (columnOf.containsKey(getCompositeKey(this.inputFiles[message.getId()].getName(), this.headerLines[message.id][column])))
        columnOf.get(getCompositeKey(this.inputFiles[message.getId()].getName(), this.headerLines[message.id][column])).addValueToColumn(
    else {
        columnOf.put(getCompositeKey(this.inputFiles[message.getId()].getName(), this.headerLines[message.id][column]), new Column(colum
        columnOf.get(getCompositeKey(this.inputFiles[message.getId()].getName(), this.headerLines[message.id][column])).addValueToColumn(
```

Checking for INDs

- We send column keys to dependency workers
- Dependency workers also have Bucket we check internal buckets
- If required, we request for needed column
- Check if both Column contain the same values



```
this.getContext().getLog().info("Looking for IND between {} and {}", column1, column2);
result = column1.getColumnValues().containsAll(column2.getColumnValues());
this.getContext().getLog().info("{}", result);
```