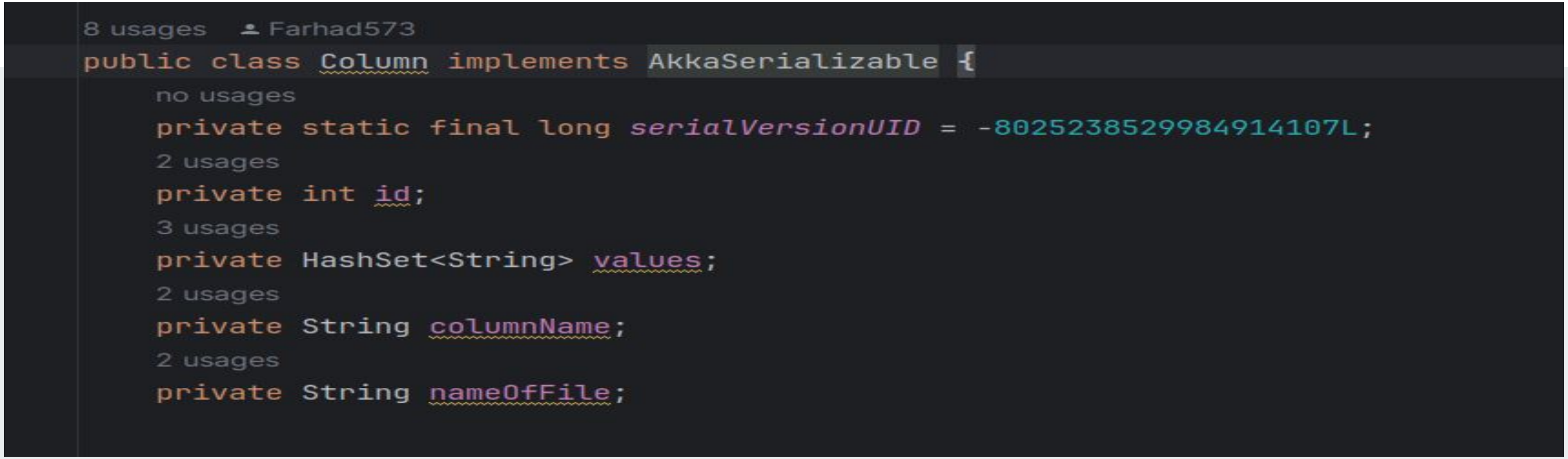
Big Data System Presentation Team: IT-Boys

Sayedfarhad Emami Dehcheshmeh

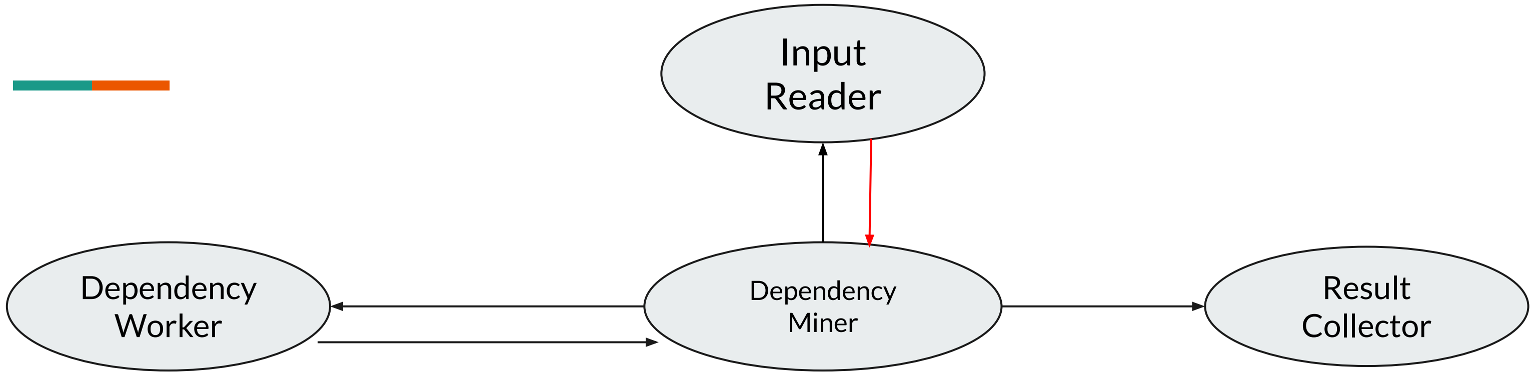
Mohsen Saleki

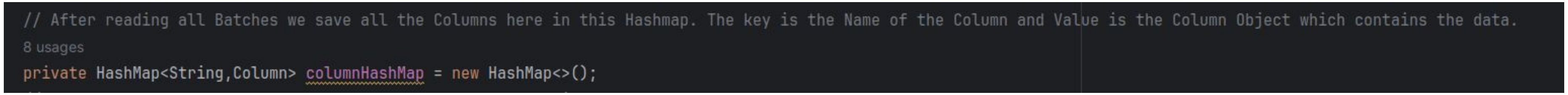
Sina Fadavi



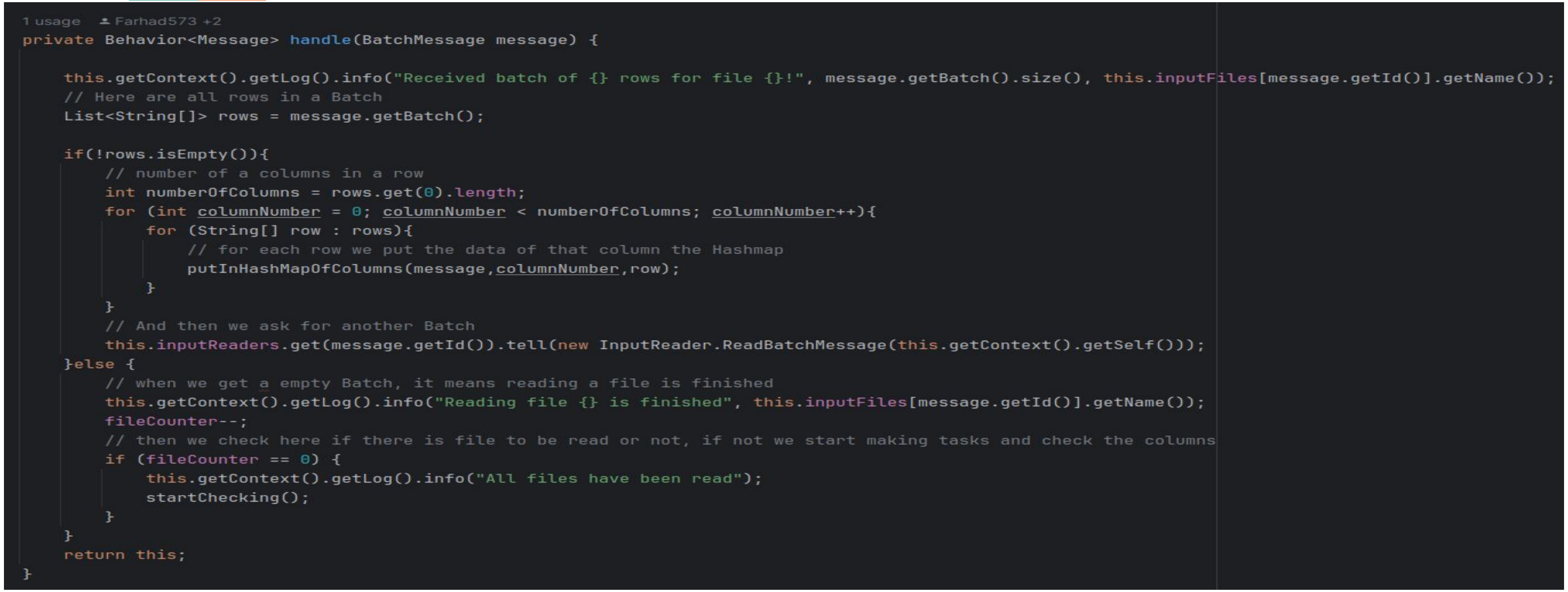
```
8 usages   Farhad573
public class Column implements AkkaSerializable {
    no usages
    private static final long serialVersionUID = -8025238529984914107L;
    2 usages
    private int id;
    3 usages
    private HashSet<String> values;
    2 usages
    private String columnName;
    2 usages
    private String nameOfFile;
```

This Class represent the Columns of our Data. All the values of each Column will be stored in the Hashset.

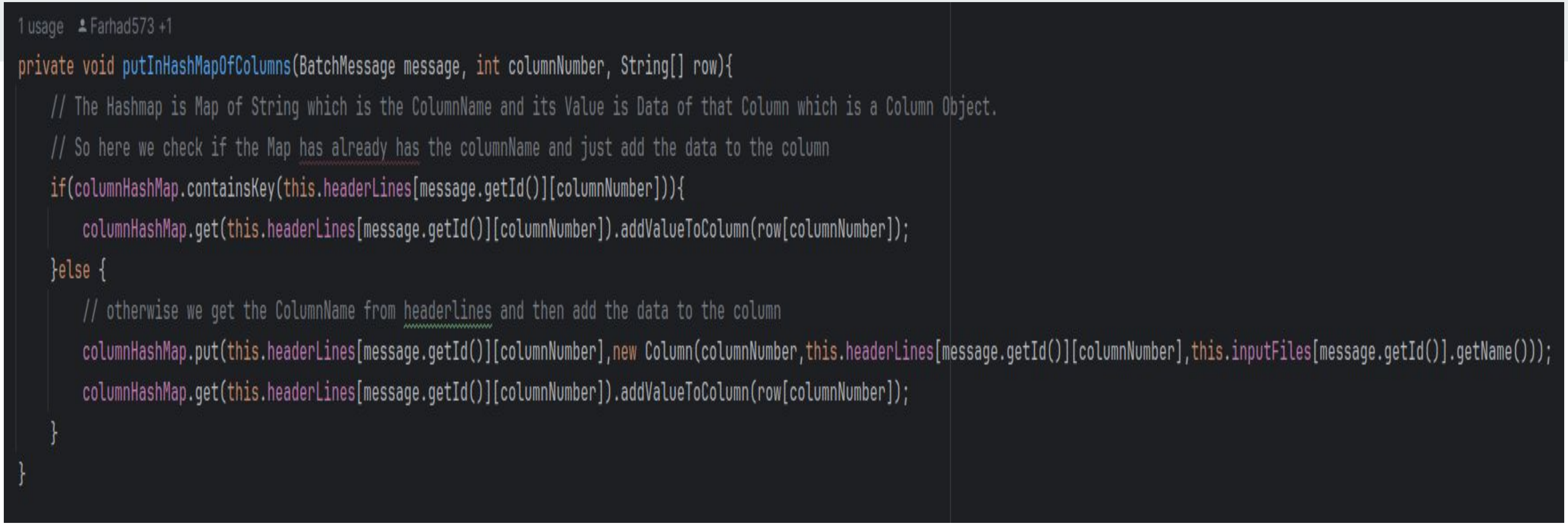

Input Reader
Dependency Worker
Dependency Miner
Result Collector



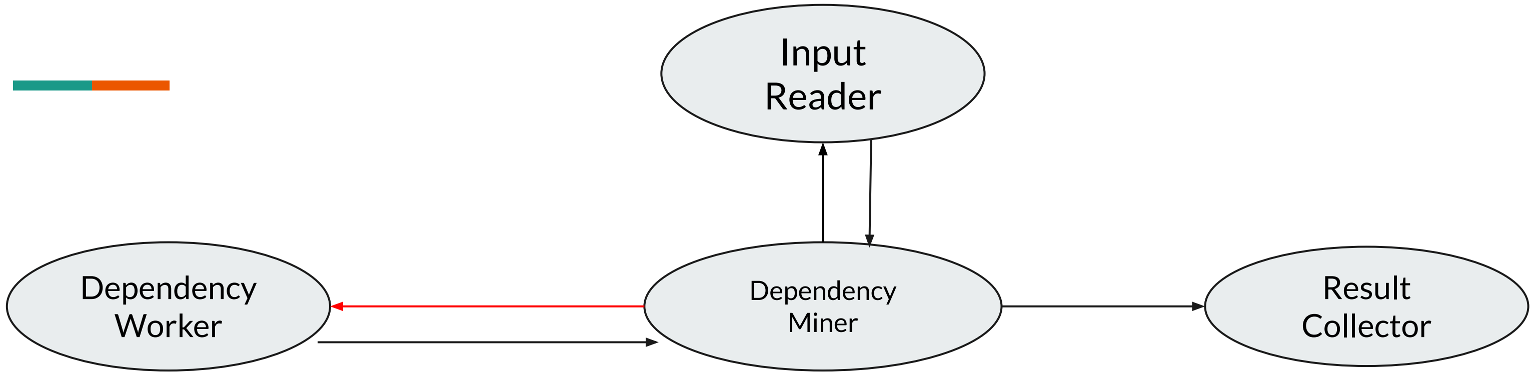
```
// After reading all Batches we save all the Columns here in this Hashmap. The key is the Name of the Column and Value is the Column Object which contains the data.
8 usages
private HashMap<String,Column> columnHashMap = new HashMap<>();
```

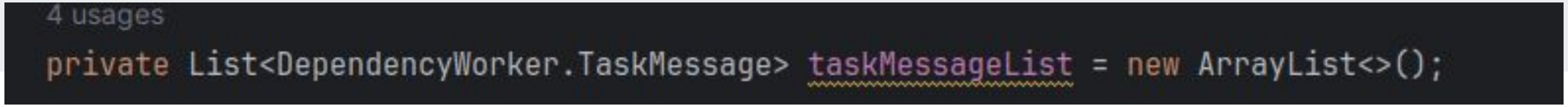


```
1 usage   Farhad573 +2
private Behavior<Message> handle(BatchMessage message) {

    this.getContext().getLog().info("Received batch of {} rows for file {}!", message.getBatch().size(), this.inputFiles[message.getId()].getName());
    // Here are all rows in a Batch
    List<String[]> rows = message.getBatch();

    if(!rows.isEmpty()){
        // number of a columns in a row
        int numberOfColumns = rows.get(0).length;
        for (int columnNumber = 0; columnNumber < numberOfColumns; columnNumber++){
            for (String[] row : rows){
                // for each row we put the data of that column the Hashmap
                putInHashMapOfColumns(message,columnNumber,row);
            }
        }
        // And then we ask for another Batch
        this.inputReaders.get(message.getId()).tell(new InputReader.ReadBatchMessage(this.getContext().getSelf()));
    }else {
        // when we get a empty Batch, it means reading a file is finished
        this.getContext().getLog().info("Reading file {} is finished", this.inputFiles[message.getId()].getName());
        fileCounter--;
        // then we check here if there is file to be read or not, if not we start making tasks and check the columns
        if (fileCounter == 0) {
            this.getContext().getLog().info("All files have been read");
            startChecking();
        }
    }
    return this;
}
```

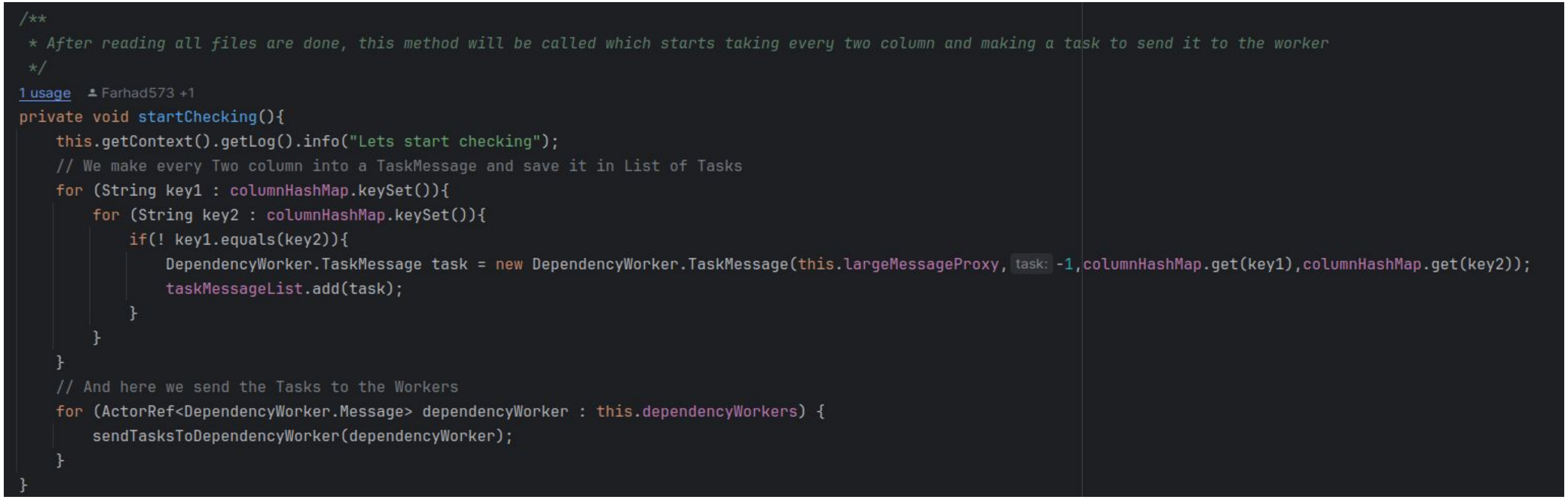


```
1 usage  Farhad573 +1
private void putInHashMapOfColumns(BatchMessage message, int columnNumber, String[] row){
    // The Hashmap is Map of String which is the ColumnName and its Value is Data of that Column which is a Column Object.
    // So here we check if the Map has already has the columnName and just add the data to the column
    if(columnHashMap.containsKey(this.headerLines[message.getId()][columnNumber])){
        columnHashMap.get(this.headerLines[message.getId()][columnNumber]).addValueToColumn(row[columnNumber]);
    }else {
        // otherwise we get the ColumnName from headerlines and then add the data to the column
        columnHashMap.put(this.headerLines[message.getId()][columnNumber],new Column(columnNumber,this.headerLines[message.getId()][columnNumber],this.inputFiles[message.getId()].getName()));
        columnHashMap.get(this.headerLines[message.getId()][columnNumber]).addValueToColumn(row[columnNumber]);
    }
}
```

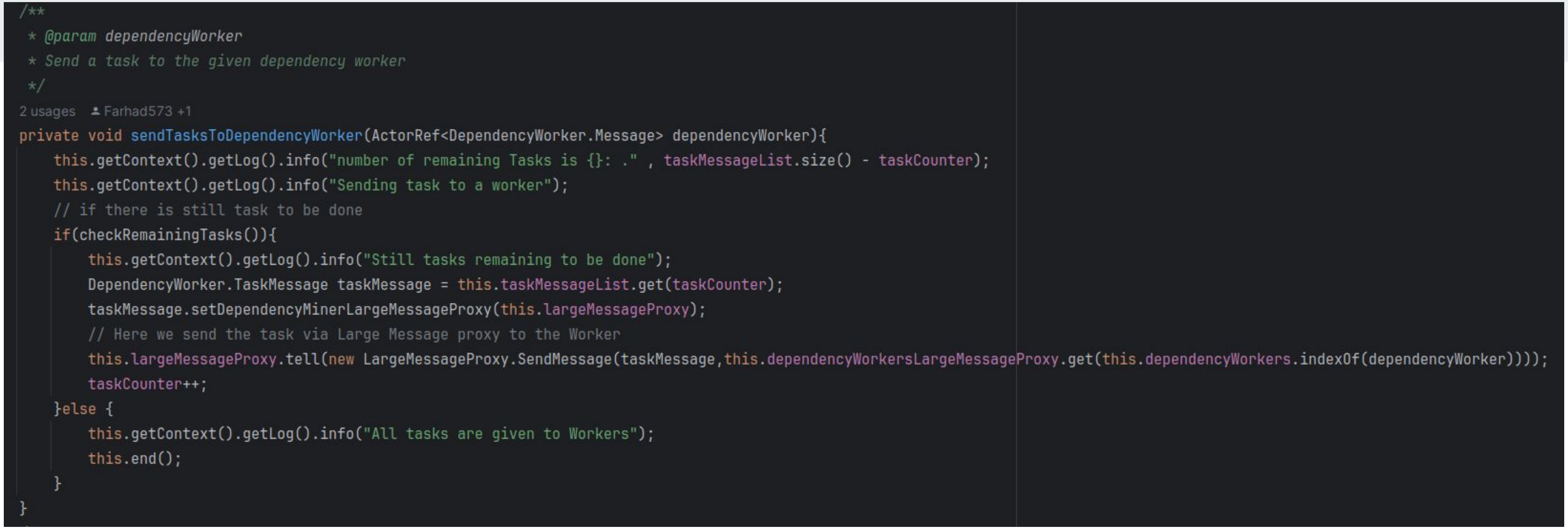

Input Reader
Dependency Worker
Dependency Miner
Result Collector

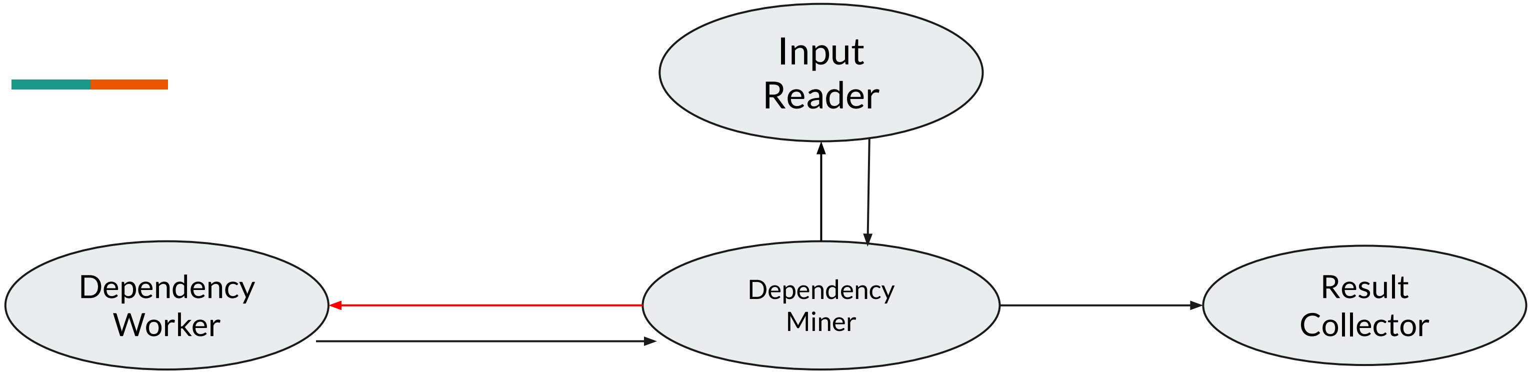


```
4 usages
private List<DependencyWorker.TaskMessage> taskMessageList = new ArrayList<>();
```

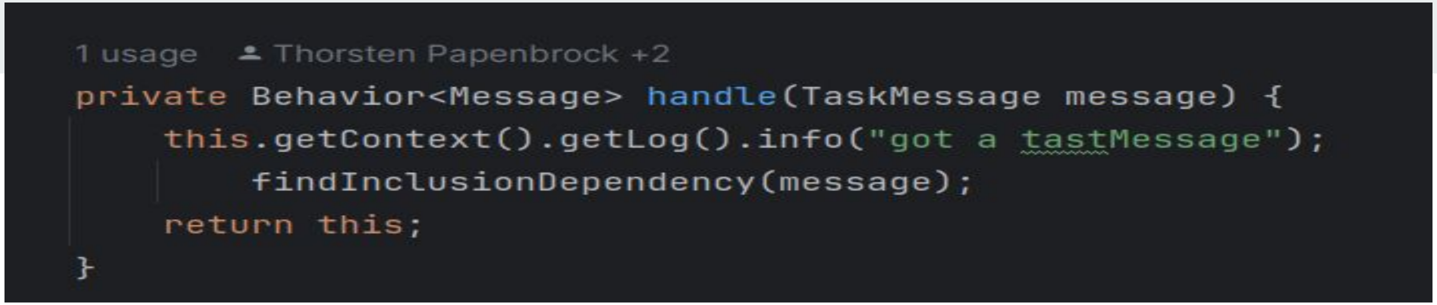


```
/**
 * After reading all files are done, this method will be called which starts taking every two column and making a task to send it to the worker
 */
1 usage  Farhad573 +1
private void startChecking(){
    this.getContext().getLog().info("Lets start checking");
    // We make every Two column into a TaskMessage and save it in List of Tasks
    for (String key1 : columnHashMap.keySet()){
        for (String key2 : columnHashMap.keySet()){
            if(! key1.equals(key2)){
                DependencyWorker.TaskMessage task = new DependencyWorker.TaskMessage(this.largeMessageProxy, task: -1,columnHashMap.get(key1),columnHashMap.get(key2));
                taskMessageList.add(task);
            }
        }
    }
    // And here we send the Tasks to the Workers
    for (ActorRef<DependencyWorker.Message> dependencyWorker : this.dependencyWorkers) {
        sendTasksToDependencyWorker(dependencyWorker);
    }
}
```

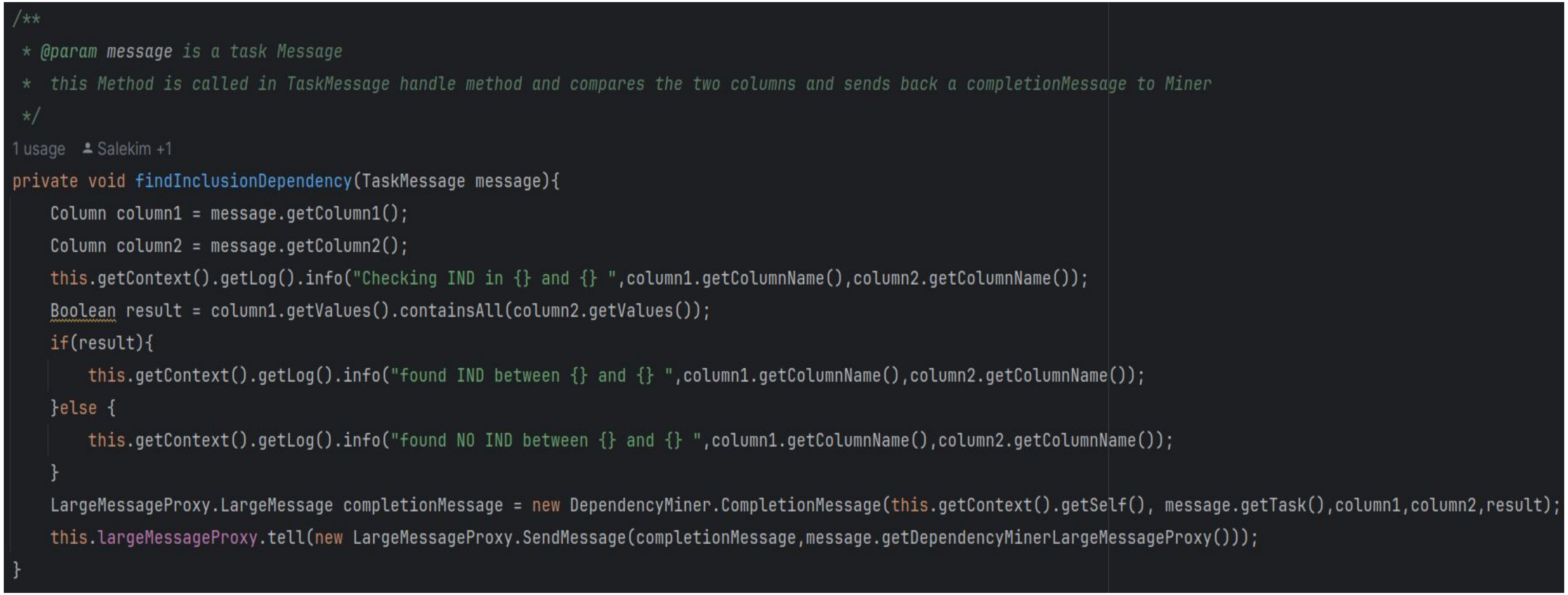

```java
/**
 * @param dependencyWorker
 * Send a task to the given dependency worker
 */
2 usages  Farhad573 +1
private void sendTasksToDependencyWorker(ActorRef<DependencyWorker.Message> dependencyWorker){
    this.getContext().getLog().info("number of remaining Tasks is {}: ." , taskMessageList.size() - taskCounter);
    this.getContext().getLog().info("Sending task to a worker");
    // if there is still task to be done
    if(checkRemainingTasks()){
        this.getContext().getLog().info("Still tasks remaining to be done");
        DependencyWorker.TaskMessage taskMessage = this.taskMessageList.get(taskCounter);
        taskMessage.setDependencyMinerLargeMessageProxy(this.largeMessageProxy);
        // Here we send the task via Large Message proxy to the Worker
        this.largeMessageProxy.tell(new LargeMessageProxy.SendMessage(taskMessage,this.dependencyWorkersLargeMessageProxy.get(this.dependencyWorkers.indexOf(dependencyWorker))));
        taskCounter++;
    }else {
        this.getContext().getLog().info("All tasks are given to Workers");
        this.end();
    }
}
```



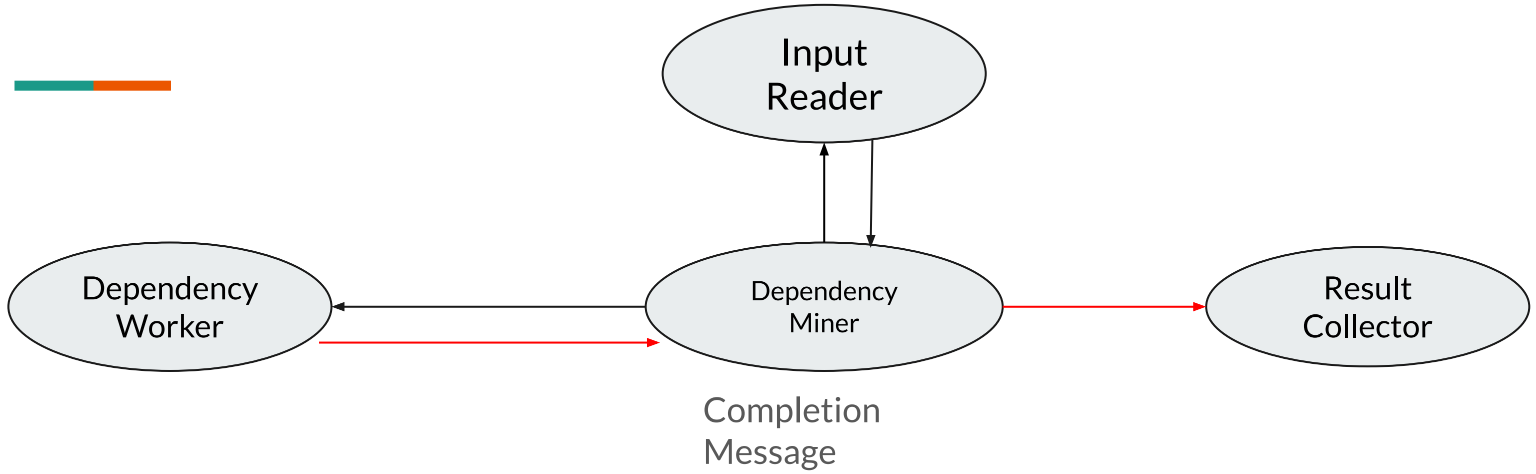
Task Message

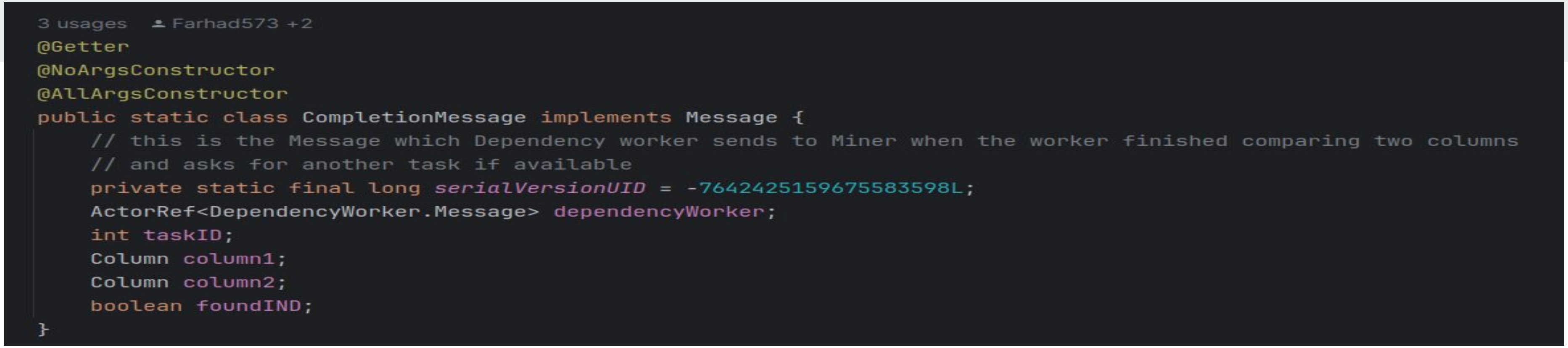


```
1 usage   Thorsten Papenbrock +2
private Behavior<Message> handle(TaskMessage message) {
    this.getContext().getLog().info("got a tastMessage");
        findInclusionDependency(message);
    return this;
}
```

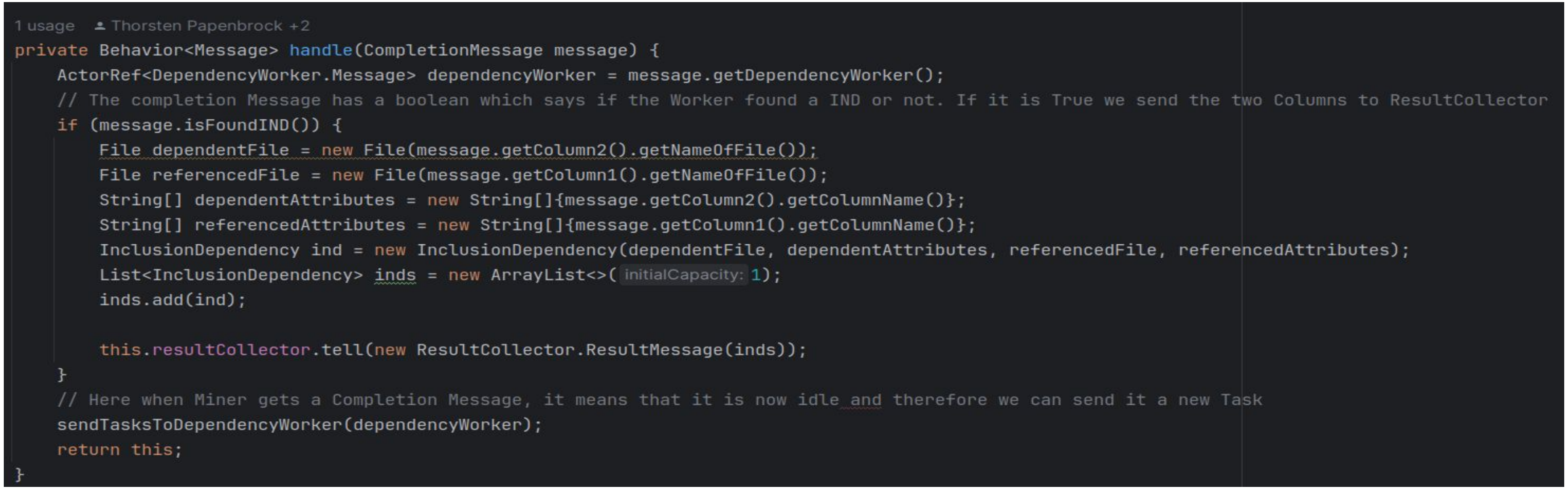


```
/**
 * @param message is a task Message
 *  this Method is called in TaskMessage handle method and compares the two columns and sends back a completionMessage to Miner
 */
1 usage   Salekim +1
private void findInclusionDependency(TaskMessage message){
    Column column1 = message.getColumn1();
    Column column2 = message.getColumn2();
    this.getContext().getLog().info("Checking IND in {} and {} ",column1.getColumnName(),column2.getColumnName());
    Boolean result = column1.getValues().containsAll(column2.getValues());
    if(result){
        this.getContext().getLog().info("found IND between {} and {} ",column1.getColumnName(),column2.getColumnName());
    }else {
        this.getContext().getLog().info("found NO IND between {} and {} ",column1.getColumnName(),column2.getColumnName());
    }
    LargeMessageProxy.LargeMessage completionMessage = new DependencyMiner.CompletionMessage(this.getContext().getSelf(), message.getTask(),column1,column2,result);
    this.largeMessageProxy.tell(new LargeMessageProxy.SendMessage(completionMessage,message.getDependencyMinerLargeMessageProxy()));
}
```


Input Reader
Dependency Worker
Dependency Miner
Result Collector
Completion Message



```
3 usages   Farhad573 +2
@Getter
@NoArgsConstructor
@AllArgsConstructor
public static class CompletionMessage implements Message {
    // this is the Message which Dependency worker sends to Miner when the worker finished comparing two columns
    // and asks for another task if available
    private static final long serialVersionUID = -7642425159675583598L;
    ActorRef<DependencyWorker.Message> dependencyWorker;
    int taskID;
    Column column1;
    Column column2;
    boolean foundIND;
}
```



```
1 usage   Thorsten Papenbrock +2
private Behavior<Message> handle(CompletionMessage message) {
    ActorRef<DependencyWorker.Message> dependencyWorker = message.getDependencyWorker();
    // The completion Message has a boolean which says if the Worker found a IND or not. If it is True we send the two Columns to ResultCollector
    if (message.isFoundIND()) {
        File dependentFile = new File(message.getColumn2().getNameOfFile());
        File referencedFile = new File(message.getColumn1().getNameOfFile());
        String[] dependentAttributes = new String[]{message.getColumn2().getColumnName()};
        String[] referencedAttributes = new String[]{message.getColumn1().getColumnName()};
        InclusionDependency ind = new InclusionDependency(dependentFile, dependentAttributes, referencedFile, referencedAttributes);
        List<InclusionDependency> inds = new ArrayList<>( initialCapacity: 1);
        inds.add(ind);

        this.resultCollector.tell(new ResultCollector.ResultMessage(inds));
    }
    // Here when Miner gets a Completion Message, it means that it is now idle and therefore we can send it a new Task
    sendTasksToDependencyWorker(dependencyWorker);
    return this;
}
```