How to write an implementation of IEngineTestData

# The implementation has to be abstract

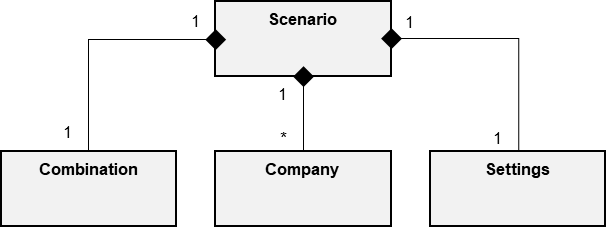
The class should only call the method ‘public boolean check(Scenario scenario)’ and not implement it.

# Implement the method ‘public boolean start()’

The purpose of the class is to generate combination of

* Import File
* Existing data (data existing on the DB before the import functionality)
* Import Settings (parameters of the import functionality)

Each combination of these data is a ‘Scenario’.



For each Scenario has to be called the method

**public** **boolean** check(Scenario scenario)

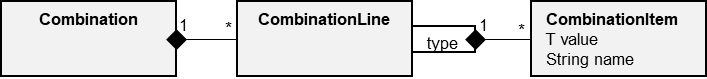
## Import File

The class does not generate the import file directly. The generation of the file is made by the Checker (not provided) checking the Scenario.combination value.

The instance of ‘Combination’ contains one or more instances of ‘CombinationLine’ (the lines of the import file). Each ‘CombinationLine’ contains one or more instances of ‘CombinationItem’ (the cell of the import file).

The CombinationItem contains a value (the value of the object) and a name (the value converted as text).

Each CombinationItem is associated to the CombinationLine by type (the type if the column).



Relation between Type and Value

|  |  |
| --- | --- |
| **Type (IFRS16ImportAssignmentType)** | **Value** |
| Contract Number | java.lang.String |
| Creditor Number | java.lang.String |
| Creditor Name | java.lang.String |
| Group Position | ConcreteAccount |
| Designation Leased Object | java.lang.String |
| Start Date of Contract | java.lang.Long |
| End Date of Contract | java.lang.Long |
| Probable End of Contract | java.lang.Long |
| Interest Rate | java.lang.Double |
| Partner Company | java.lang.String |
| Condition Type | IFRS16ConditionType |
| From Date | java.lang.Long |
| Until Date | java.lang.Long |
| Cost Center | java.lang.String |
| Payment Cycle | IFRS16PaymentCycle |
| Payment Date Type | IFRS16PaymentDateType |
| Amount Without Value Added Tax | java.lang.Double |
| VAT Rate Type | IFRS16VATRateType |

Use one of the ‘static’ methods of the class CombinationItem to create an instance of a ‘valid’ CombinationItem. These methods format automatically the ‘name’ field.

Use the method ‘public static CombinationItem<String> getNewCombinationItem(String value)’ to create ‘wrong’ CombinationItem

Example of Combination

|  |  |  |  |
| --- | --- | --- | --- |
|  | **IFRS16ImportAssignmentType**  **CREDITORNUMBER** | **IFRS16ImportAssignmentType**  **STARTDATEOFCONTRACT** | **IFRS16ImportAssignmentType**  **INTERESTRATE** |
| **CombinationLine 0** | **CombinationItem (valid)**  Value=Abc  Name=Abc | **CombinationItem (valid)**  Value=java.lang.Long(1523577600000)  Name =13.04.2018 | **CombinationItem (valid)**  Value=java.lang.Double(1.45)  Name =1.45 |
| **CombinationLine 1** | **CombinationItem (wrong)**  Value=  Name = | **CombinationItem (wrong)**  Value=null  Name =99.99.999 | **CombinationItem (wrong)**  Value=null  Name =Abc |
| **CombinationLine n** | […] | […] | […] |

Transformation of the Combination to an Import File (csv)

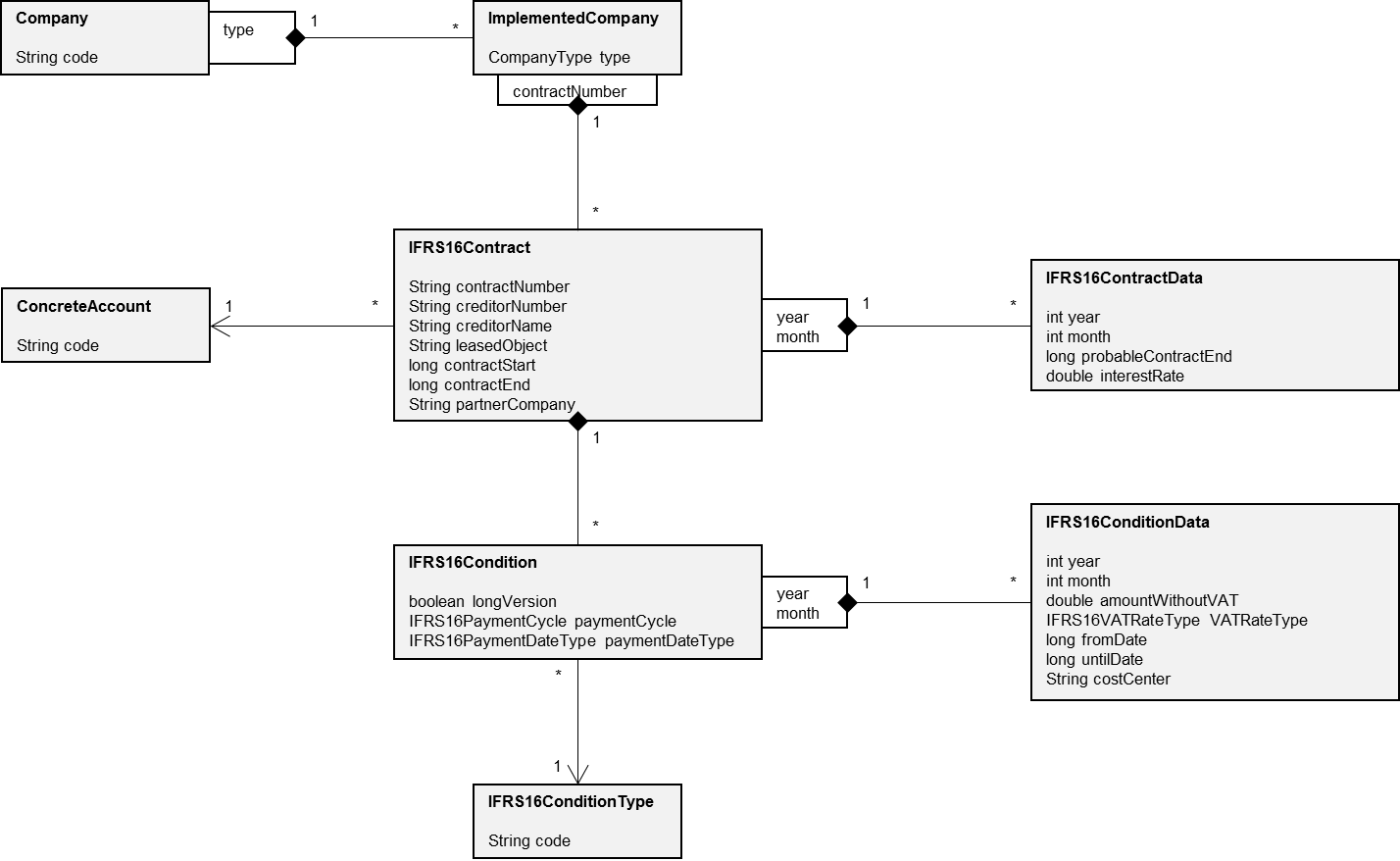
Line0: “Abc”,”13.04.2018”,”1.45”

Line1: “”,”99.99.9999”,”Abc”

The combination of the type of the CombinationItem has to be valid and the same for each CombinationLine. Different combination of the type have to be created in different Combination

## Existing Data

The class not generates data on the DB directly. The generation of the data is made by the Checker (not provided) checking the Scenario.companyList value.



The ‘ConcreteAccount’ and ‘IFRS16ConditionType’ does not have to be generated. They are part of the structure of the software. Use the following methods of the ‘Util’ class to obtain the list of the objects:

**public** **static** HashSet<ConcreteAccount> getAllConcreteAccount()

**public** **static** HashSet<IFRS16ConditionType> getAllIFRS16ConditionType()

The Company and ImplementedCompany data have always to be created directly.

**for**(String code : …) {

Company company = **new** Company();

company.setCode(code);

companyList.add(company);

**for**(CompanyType companyType : **new** CompanyType[] {CompanyType.***PRODUCTIVE***, CompanyType.***CONSOLIDATED***}) {

ImplementedCompany implementedCompany = **new** ImplementedCompany();

implementedCompany.setType(companyType);

company.createChildImplementedCompany(implementedCompany);

}

}

There are two ways to generate the IFRS16Contract data: directly or parsing a CombinationLine instance

IFRS16Contract ifrs16Contract = **new** Checker().createIFRS16Contract(settings, combinationLine);

**if**(ifrs16Contract != **null**) {

implementedCompany.createChildIFRS16Contract(ifrs16Contract);

}

## Import Settings

The ‘ImportSettings’ data are used by the Checker (not provided) during the import checking the Scenario.settings value.

### Locale

Is the ‘Locale’ chosen by the user on log-in. The available Locale are the following:

* Locale.GERMAN
* Locale.ENGLISH

The format of the ‘Name’ attribute of the ‘CombinationItem’ instances depends on this field

### ImplementedCompany, LongVersion

The ‘ImplementedCompany’ chosen as target of the import functionality (imported IFRS16Contract data will be aggregated to this ImplementedCompany). The ImplementedCompany has to exist.

The ‘LongVersion’ Boolean field indicates the version of the Import File.

### Year, Month

The time-dimension chosen as target of the import functionality (IFRS16ContractData and IFRS16ConditionData). Year and Month [0-11] ([Calendar.JANUARY, Calendar.DECEMBER]) have to be valid.

### FilePath

The absolute path and name of the Import File. The Import File will be written in this position before the import. On test fail the Import File will be not deleted.

### Name

The name of the Settings

### DigitGroupingSymbol, DecimalSeparator

The symbols chosen to format the Double values (Interest Rate and Amount Without Value Added Tax).

The available DigitGroupingSymbol (thousand separator) are the following:

* DigitGroupingSymbol.NULL (none)
* DigitGroupingSymbol.POINT (.)
* DigitGroupingSymbol.COMMA (,)
* DigitGroupingSymbol.NO\_BREAK\_SPACE (char 160)
* DigitGroupingSymbol.SINGLE\_QUOTATION\_MARK (‘)
* DigitGroupingSymbol.NORMAL\_SPACE ( )

The available DecimalSeparator are the following:

* DigitGroupingSymbol.POINT (.)
* DigitGroupingSymbol.COMMA (,)

The chosen DigitGroupingSymbol and DecimalSeparator have to be valid and do not have to be the same.

The format of the ‘Name’ attribute of the ‘CombinationItem’ instances depends on these fields.

### DateFormat

The symbols chosen to format the Date fields (Start Date of Contract, End Date of Contract, Probable End of Contract, From Date, Until Date).

The available DateFormat are the following:

* Constants.SETTTING\_DATE\_DE
* Constants.SETTTING\_DATE\_EN

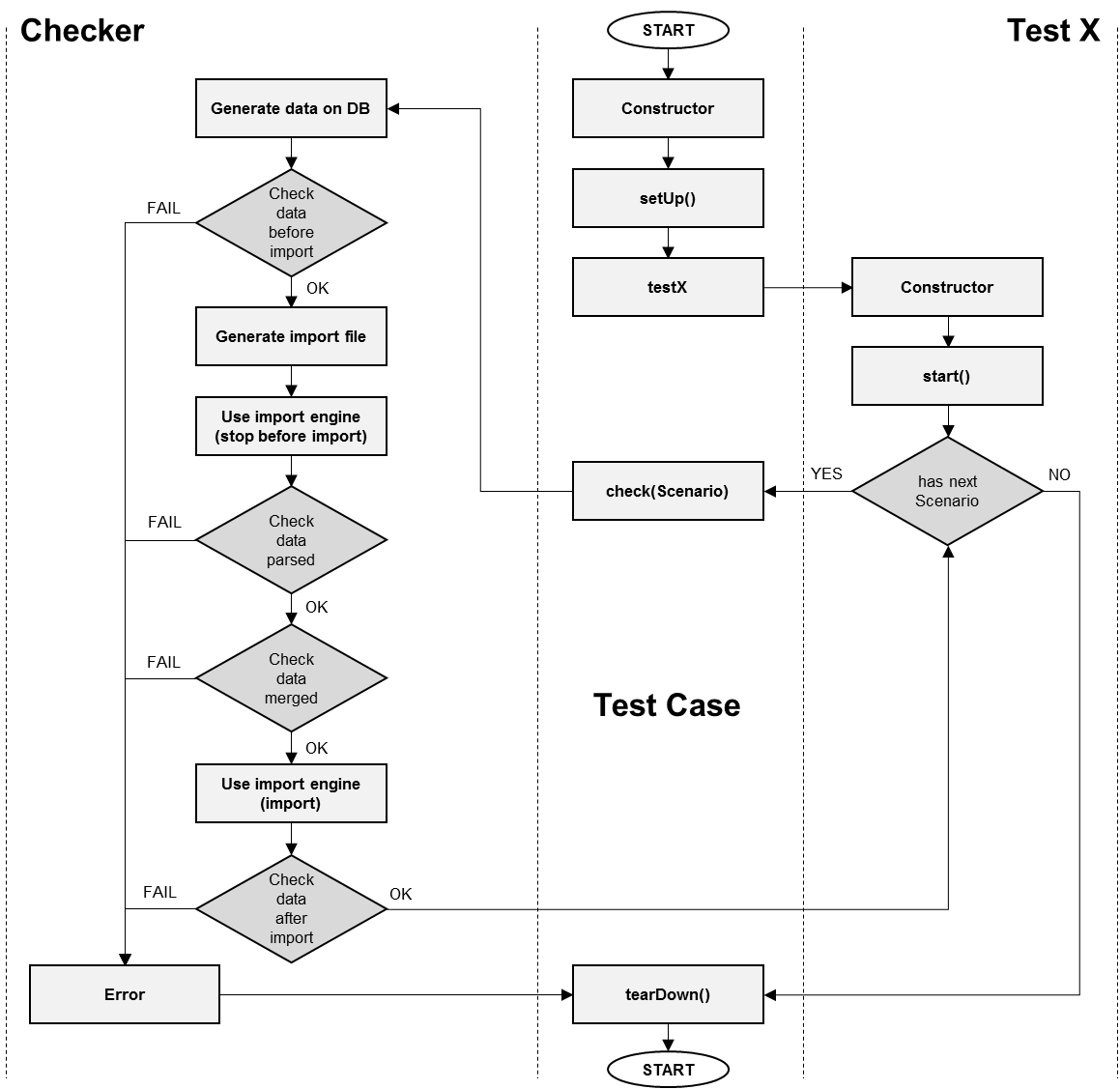
The format of the ‘Name’ attribute of the ‘CombinationItem’ instances depends on this field.

### TypeList

The list of the type (IFRS16ImportAssignmentType) used in the Combination.

The sorting of the Import File columns depends on this field.

# Test Case (not provided)



**public** **class** EngineTest **extends** AbstractCISTest {

**public** EngineTest(String testCaseName) {

**super**(testCaseName);

}

**public** **void** testSample() {

**new** SampleEngineTestData() {

@Override

**public** **boolean** check(Scenario scenario) {

checkImport(scenario);

**return** **true**;

}

}.start();

}

**private** **void** checkImport(Scenario scenario) {

System.***out***.println("START...");

**long** time = System.*currentTimeMillis*();

// Data generation

**new** ATransactionOperation<Boolean>(**this**, **false**) {

@Override

**protected** Boolean operation() **throws** PUserException {

DataGenerator.*generate*(

scenario.getCompanyList(),

scenario.getSettings().isLongVersion(),

getDbController()

);

**return** **true**;

}

}.execute();

String errorMessage = **new** ATransactionOperation<String>(**this**, **false**) {

@Override

**protected** String operation() **throws** PUserException {

CisController controller = getDbController();

IFRS16ImportCompanyDataEngine engine = **new** IFRS16ImportCompanyDataEngine(controller);

ImportChecker checker = **new** ImportChecker();

// Check before import

String errorMessage = checker.checkBeforeImport(scenario, engine, controller);

**if**(errorMessage != **null**) {

**return** errorMessage;

}

// File generation

**try** {

FileGenerator.*generate*(scenario);

}

**catch**(Exception exception) {

exception.printStackTrace();

**return** "An exception occurred during the import file generation!";

}

// Use panels

**try** {

Importer.*usePanels*(scenario, engine, controller);

}

**catch**(Exception exception) {

exception.printStackTrace();

**return** "An exception occurred during the engine use!";

}

// Check data parsed

errorMessage = checker.checkDataParsed(scenario, engine, controller);

**if**(errorMessage != **null**) {

**return** errorMessage;

}

// Check data merged

errorMessage = checker.checkDataMerged(scenario, engine, controller);

**if**(errorMessage != **null**) {

**return** errorMessage;

}

// Import

**try** {

Importer.*doImport*(engine);

}

**catch**(Exception exception) {

exception.printStackTrace();

**return** "An exception occurred during the import!";

}

// Check after import

errorMessage = checker.checkAfterImport(scenario, engine, controller);

**if**(errorMessage != **null**) {

**return** errorMessage;

}

**return** **null**;

}

}.execute();

AbstractULCTest.*assertTrue*(errorMessage, errorMessage == **null**);

System.***out***.println("...END (" + (System.*currentTimeMillis*() - time) + " ms)");

}

}

The lines in yellow will be replicated for each group.