```
class FireAlarm is subclass of Alarm

end FireAlarm
```

The last class `FireAlarm` is an empty class that inherits all its content from the `Alarm` class.

A UML file is then generated from the VDM model, using the VDM2UML functionality and the resulting PlantUML notation is constructed using the XML2PlantUML table.

```
class Plant {
    -schedule : Period
}
class Alarm {
    -descr : String
    +setDescr() : String
}

class FireAlarm {
    -descr : String
    +setDescr() : String
}

Plant -right- "0..*" Alarm : alarms
Alarm <|-- FireAlarm
```

Which can then be exported as a PlantUML diagram, shown ~~below~~ in figure 7

5 Dependency of UML Transformations on the Eclipse IDE

Our method of implementing the UML transformations hinge on packaging JAR files that contain the desired main functions as well as all the necessary dependencies for it to function. Doing so will make the plugin future proof, since no matter if the Java environment or the Eclipse package repositories are updated, the JAR will contain all dependencies necessary to run that specific main function. However, getting Maven to package these JARs correctly requires some considerations.

The VDM to UML transformation is implemented in Overture on the Eclipse side. However, contrary to eg. the java code generation, the UML transformations are implemented as plugins on the IDE side of the Overture Tool. The plugins being dependent on the GUI aspects of Eclipse makes it incompatible with jar packaging through Maven. This is because the project has dependencies that are specific to the Eclipse IDE as well as Eclipse dependencies that Maven has trouble finding because they have in some way become obsolete.