Packages and their Dependencies

Mikael Svahnberg*

2016-04-27

1 Introduction

I received a question:

> Hej! Vår grupp har stött på ett problem med Uppgift 2, kapitel 2.2: Packages and their relations/dependencies. Vi vet inte riktigt vad som ska göras här och vore tacksamma om vi fick ett exempel.

This made me realise that this is an aspect that I have not covered in the course. Packages are covered in chapter 31.18 in the course book (with some guidelines for how to think when creating packages in chapter 35.1), but here is the TL;DR of them.

2 What are Packages?

A package is a means to hide lower-level details. For example, even if you model each sub-system in your application on a separate page, you will then need to have an overview of how your separate pages fit together. When you do architectural design (in another course), you will (for one of the views of your system, the Module View) use packages to illustrate how components are implemented as one or a small set of packages.

You can also use them to illustrate that some classes (or other packages) logically "belong together". If we look at the Dictionary example discussed in lecture 07 (From Design to Code), we can use three high-level packages to illustrate that we have a Model, Views, and a Controller. In this simple example, there are only a few classes in each, but in a more complex system, you may have quite a number of collaborating classes in each package.

The notation is simply a folder, and this is the analogy as well – that it is a folder where you may hide stuff. You would typically use this "hide-the-inside"-mode when you are showing a high-level view of the system. Later on, you would have to open the folder and display the insides as well.

2.1 Example: Package of Packages EXAMPLE

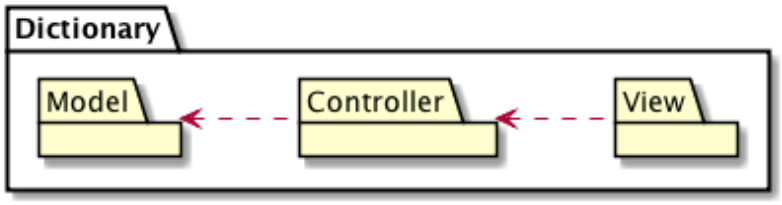
In this example, I am – on a very high level – modelling the model, view, and controller packages of the Dictionary system.

*Mikael.Svahnberg@bth.se

I have added dependencies between the packages as well. *Please Note* the direction of the dependency arrow! In this example, *View* depends on *Controller*, and *Controller* depends on the *Model* package.

Source (plantuml) and Diagram:

```
@startuml
left to right direction
package Dictionary  {
package Model
package View
package Controller
Model <.. Controller
Controller <.. View
}
@enduml
```

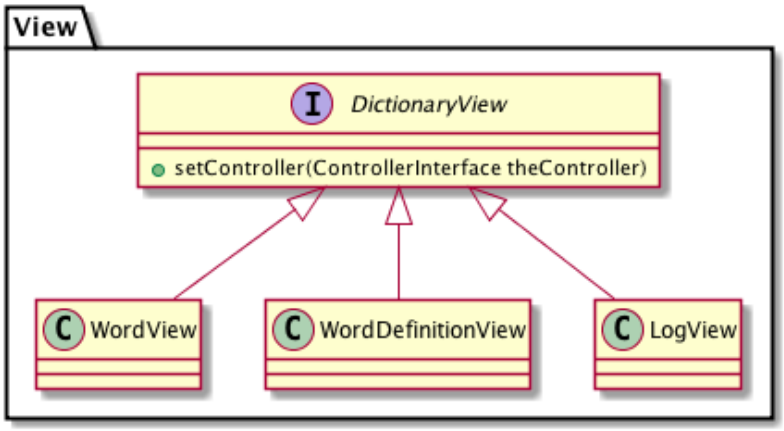


2.2 Example: Package of Classes EXAMPLE

A first look inside the *View* Package:

Source and Diagram:

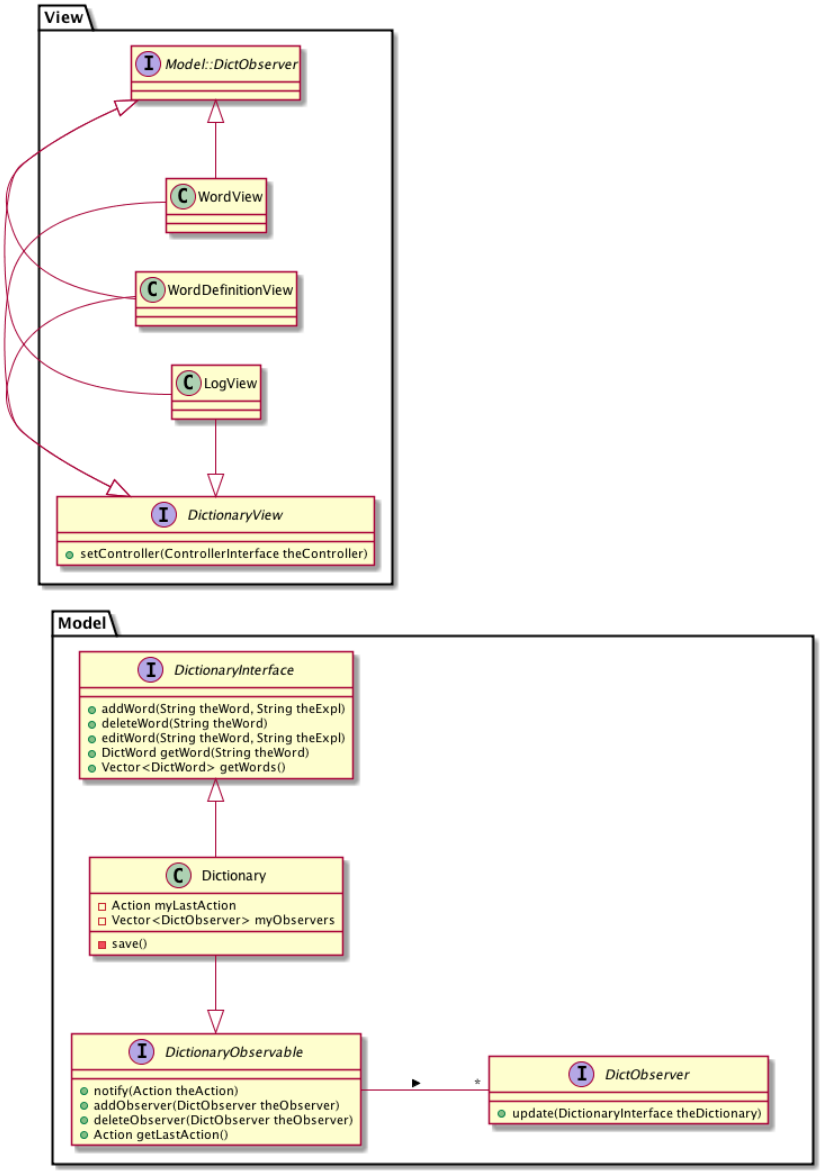
```
@startuml
Package View {
interface DictionaryView
DictionaryView : +setController(ControllerInterface theController)

DictionaryView <|-- WordView
DictionaryView <|-- WordDefinitionView
DictionaryView <|-- LogView
}
@enduml
```



2.3 Example: Referring to elements from another package

Example

A package "owns" everything that is inside the package. This means that if we want to refer to something from another package, we need to have a way to "import" it – or link directly across package borders. Both are possible, but the "import" facility is preferred since it leads to a less cluttered diagram. This is done using the *scope* operator from C++: `Package::Class` .


View
Model::DictObserver
WordView
WordDefinitionView
LogView
DictionaryView
setController(ControllerInterface theController)
Model
DictionaryInterface
addWord(String theWord, String theExpl)
deleteWord(String theWord)
editWord(String theWord, String theExpl)
DictWord getWord(String theWord)
Vector<DictWord> getWords()
Dictionary
Action myLastAction
Vector<DictObserver> myObservers
save()
DictionaryObservable
notify(Action theAction)
addObserver(DictObserver theObserver)
deleteObserver(DictObserver theObserver)
Action getLastAction()
*
DictObserver
update(DictionaryInterface theDictionary)

2.4 One More Example: Pacman EXAMPLE