Student information

Cursuscode IM0102

Scenario JabberPoint with themes

Name A. Slomp

Student number 838768442

Name W. van der Weij

Student number 851988675

1 Introduction

This is our report of the final assignment of the Design patterns course of the Open Universiteit. In this report we describe our solution to the redesign of the JabberPoint application.

This report is structured as follows. The report starts with a description of the way we worked together on the assignment. The problem analysis describes the goal, concepts with commonality and variability analysis, application functionality, rules, actions and the assumptions during the assignment.

Afterwards we gave of view of how we implemented design for change. How we applied the guidelines and principles. We outlined the package structure as well the programming guidelines. In the design patterns we described which patterns we used and how we applied them on the code. Later in the process we found a lot questions unanswered or answered but explicit choices needed to be made. We described those in the choices. There are also questions pertaining to the scope of the project which we handled there. Finally we described the sourcecode and supported them with class diagrams in the latest chapter.

2 Project execution

Here we describe how we handled the assignment together. The numbers indicate the explicit ordering of the activities.

1. We both studied the assignment separately to prepare for our first meeting.

2. After a couple of days we met at Wilkos place for the first time to discuss the assignment. Together we went through the assignment, the program code, the running application and we discussed possible solutions on conceptual level as well on implementation level. Just exchanging ideas, brain storming and discussing about possible solutions. We also discussed the tools to use (Eclipse with Papyrus plugin) and the format of the report (Latex). Furthermore we looked together at the git repository, the usage of github.com and how to operate it locally. We also discussed the planning of the assignment.

3. We both started working on the assignment separately. We both agreed to do most of the work separately, but based on our discussions and compare solutions. From our discussions we decided to have a different focus, but still do the work both. Arend focusing on the class diagrams already and Wilko focusing on the problem analysis.

4. About a week after the first meeting, we met again to discuss our progress. This time at Arends place. We still had a lot to discuss. We went through the code, annotating it with TODOs at places with strange code or remarkable design. We went through two more documents from you learn: *Guidelines redesign* and *Beoordeling Design patterns*. The work to be done seemed more than the expectations we had after the first meeting. Again we agreed to do both the work with focus on different items. Arend puts together both reports and looks at the design, while Wilko does the CVA, choices and description of design patterns to be used.

5. We both continued work on the assignment separately, but together. Thinking and about the problem analysis and the design, drawing diagrams, looking at code and already cleaning up code.

6. Three days before the deadline we met again. We discussed our progress and the priorities of the remaining work.

7. We divided the work again and started polishing things up. We had to put an amount of work to document all the notes and diagrams we had written and drawn during our discussions. Tools to draw UML diagrams are handy and even indispensable, but during discussions and brainstorming a paper and pen are superior.

3 Problem analysis

The assignment is to redesign the existing application JabberPoint to support multiple themes. The feature request is to let the user pick one of the available themes and apply the new styles to the content of the presentation. The selected theme should be applied to the current application and the current slide.

Besides the addition of the feature request, the application is redesigned further. The goal is to improve the quality of the application overall and to achieve a better user experience, increased flexibility and easier maintainability.

Out of scope are at this moment:

- editing presentations
- editing themes

3.1 Application functionality

The JabberPoint application is started with the internal demo presentation or a presentation loaded from a file specified by the command line parameter. The first slide of the presentation is displayed. JabberPoint accepts one optional parameter during startup: a presentation file. An invalid parameter value results in an application without any presentation loaded.

The view of the application consists of a program window with a menubar at the top. The application is operated by a pointing device and a keyboard.

Menu items and key assignments:

- Open goal unclear, should let the user open a presentation?
- New clears the current presentation
- Save saves the presentation (writes theme)

- Exit Exits the application. Key: q or Q

Slide actions:

- Next slide show next slide. Keys: <page down>, <arrow down>, <enter> or '+'
- Previous slide show previous slide. Keys: <page up>, <arrow up> or '-'
- Goto slide show the slide with specified number. menu, shortcut

Help actions:

- About show a popup with information about JabberPoint.

New action:

- Select theme Let the user select a theme.

The presentation content could be read from a file or otherwise a demo presentation from within the application is read

JabberPoint does not support editing of presentations.

3.2 Concepts

A *presentation* consists of a number of sequential slides, which a user can step through forward and backwards. The presentation displays information to the audience during a talk or speech. In the context of this document a presentation is the content that the JabberPoint application shows and it is not the speech or talk to an audience itself.

A *slide* is a page in a presentation that contains display elements like text and graphics. The display elements are displayed according to predefined styles. Every slide has a title.

A slide contains *item*s such as text and images. All these items have varying and some overlapping behaviours, such as: font type, text size, background colour and size. The items on a slide have a indentation, which is also related with the appearance or style of the item.

*Theme*s arange the displaying of the elements of a presentation. A theme consists of styles that control the display behaviour such as background colour, text colour and font size. The styles within the theme are stacked and the levels are leading when a theme is applied to a slide.

A *style* prescribes the displaying behaviour of items on slides.

The *view* of the application consists of a window that has a menubar at the top left. The remainder of the window is used to render the slides of a presentation according to the styles that are defined for that side.

The application is able to read a XML file that contains a presentation according to a defined structure.

3.3 Rules

In the problem domain there are rules the concepts adhere to:

- Depending on the startup parameter a demo *presentation* or a presentation read from disk is shown.

Commonality	**Variability**	**Knows**	**Can**
Presentation	One of a kind	Title, theme, slides, current slide	Step through slides.
Slide	One of a kind	Title, items on slide	None
Items on slide	Text, image	Level, specific properties	None
Theme	One of a kind	Name, styles and background color	None
Style	One of a kind	Indent, line spacing, font properties	None

Tabel 1: Things

- The *level* of a *style* in a *theme* matches the *level* of a n*item* on a *slide*.
- The *items* on a *slide* are displayed from top to bottom.

3.4 Actions

Action	**Description**
Open	Goal within the application unclear, expected to open a *presentation*.
Save	Stores the *presentation* with the currently selected *theme*.
Exit	Ends the *application*.
Next	Step to the next *slide* of the *presentation*, if possible.
Previous	Step to the previous *slide* of the *presentation*, if possible.
Go to	Step to a specific *slide* of the *presentation*, if possible.
Select theme	Select a *theme* to apply to the current *presentation*.
Show slide	Draw the current *slide* with *item*s according to the correct *style*s.

Tabel 2: Actions

3.5 Assumptions

Assumption is that altering the supported file format is allowed, as long as the application is backwards compatible.

Assumption is that bug fixes are appreciated as long as functionality is not decreased. The behaviour of the next, previous and goto functionality is buggy. Several scenario's result in failures. For example the goto functionality does not validate input, resulting in errors and faulty behaviour for non-numerical and out of range values. Hitting next or previous slide quickly after each other let the presentation disappear (in MacOS with the menu options shortcuts).

3.6 Responsibilities

The responsibilities of conceptual classes and interfaces are described here. None of the domain classes in the model package have the responsibility to know how to display themselves. Displaying is the responsibility of the classes in the view package.

`Presentation` is responsible to know it's title, Theme, Slides and current slide number. It can let someone step through the slides or select a slide, managing the current slide number.

`Slide` is responsible to know it's title and it's items. It can do nothing more than that.

`SlideItem` is responsible to know its level. It can do nothing more than that.

`Theme` is responsible to know the styling information for a presentation. It can not do anything else. It holds the styles, the slides it applies to, the items to add to slides and background colour. It can do nothing more than that.

`Style` is responsible to know it's indent, font, font colour, font size and leading.

4 Design for change

We redesigned the JabberPoint application to handle changes well. We all know changes will come. We tried to accommodate change on all levels from software design and architecture to programming code. In this section we elaborate on these levels.

4.1 Guidelines and principles

Design decisions during the design process are guided by design principles and guidelines that we describe briefly. We have a couple of design principles:

- Single responsibility principle. Do one thing and do it well, without side effects. Separated responsibilities lead to high cohesion. A class should be responsible for itself.
- Open-closed principle. Open for extension, closed for modification. Make future changes without breaking existing classes.
- Don't repeat yourself principle. There must be a lack of redundancy. Rules should be implemented once. One rule, one place.
- Liskov substitution principle. Subtypes should be substitutable for base types. Well designed inheritance.

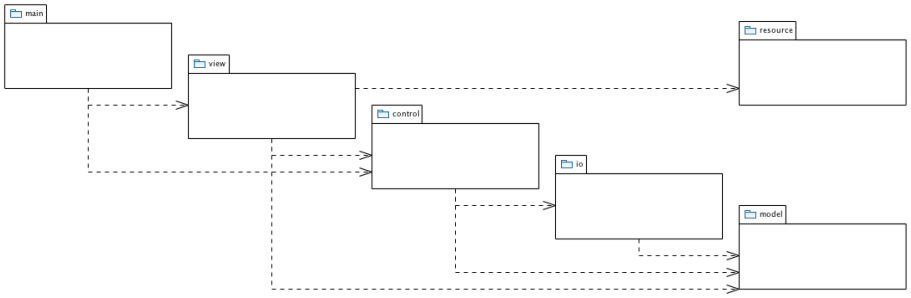
Other guidelines or best practices are:

- Prefer delegation over inheritance.
- Program to interfaces or abstract classes.
- Each interface should have one reason to change.

These principles and guidelines should lead to clarity, high cohesion and low coupling. These are ingredients for a flexible and easy to extend application. We think that is exactly what JabberPoint needs to be.

4.2 Package structure

We designed the package structure according to the package-by-layer style (see choices). The package dependencies are hierarchical and non-circular. In the package diagram one can see all arrows are one way dependencies.



Figuur 1: Package diagram

4.3 Programming guidelines

At the code level we improved the JabberPoint application significantly. A list of optimisations :

- Incorporate general code standards.
- Naming conventions for variables, constants, methods, classes and packages
- Access modifiers for constants, variables and methods
- Javadoc when necessary
- Removed duplicate constants, variables, methods and code.

We expect it would be even better if we had incorporated specialised tools to check things such as the code style, the structuring and bugs.

4.4 Design patterns

4.4.1 Model-View-Controller (MVC)

The MVC pattern separates the model, the controller and the view.

- The model manages the data, logic and rules of the application.
- The controller accepts input and converts these to commands for model or view.
- The view is responsible to display the information from the model to the user and respond to user actions.

Especially removing the view dependency from the model promotes flexibility of this design pattern. This prepares for changing or adding views without changes in the model.

4.4.2 Observer pattern

The Observer pattern decouples the JabberPoint model from the view. This is part of the MVC pattern. The view is registered as a listener, which is notified when the model changes. In the model the class Presentation is the Observable and the Observer is the SlideViewComponent class.

4.4.3 Factories

The design consists of factories to have a single point for object creation for each type of object. This supports the *program to interface* idiom, because the factories create the concrete implementations, while the application only depends on the abstract concepts. The factory classes are named XxxxFactory and are put in the factory package to make them recognizable. Currently we have factories for the classes SlideItem, Slide, Presentation, Style and Theme. With the factories we provide an single way to instantiate the classes needed.

4.4.4 Strategy pattern

The strategy pattern facilitates the varying drawing behaviour of the slide items. The ItemDrawing interface defines the methods needed to draw slide items. The implementation of the draw function is executed according to the TextDrawing or ImageDrawing strategy. If in the future another type of slide item is introduced, the ItemDrawing interface is implemented for the new algorithm. We looked at the composite pattern as well. SlideViewComponent has draw functionality and has multiple slide items such as text and images with draw functionality as well. The SlideViewComponent would be the composite. However, we judged this pattern as an overkill, because the structure is not a real tree. We already know where the leaves are and there is no need foreseen to interchange leaves with composites (slides in slides).

4.4.5 Decorator pattern

The decorator pattern allows the Theme to have subthemes. Subthemes in this case are themes which are specific to other slides. The factory will instantiate an Theme using the XML element of the file. From this factory there will be looked for slidethemes. If there is an slidetheme found, it will also be instantiated by this factory. The result will be stored in the Theme using a map, which has the specific slidepage where the theme applies to and the theme itself. By doing so it will be possible to get the theme of an specific page. To use the slidetheme or main theme there is an function which allows to get a specific slidetheme or the main theme of the presentation.

5 Choices

We embrace iterative development and work time-boxed. To respect the given deadline does not change our goal to deliver quality, but the application will not be perfect. This is a choice. We are convinced that we made great steps forward in the design and implementation of the JabberPoint implementation.

We redesign the application according to the *Redesign guidelines* document to achieve better quality in design and implementation. This choice results in better maintainability and more flexibility. We see this as an investment in future developments that will prove prosperous in the end. The alternative was to just implement the theme functionality while retaining the bad design and implementations. We could do it even faster and much uglier, without any satisfaction to end the users, the product owners and ourselves.

Bugs are fixed. This choice refers to clear bugs that can be fixed without big risks of introducing unexpected behaviour. Quality of the application and user experience are important. The alternative of leaving all the bugs in was just no option to us.

Given the major redesign and refactoring we decided to label the application version 2.0. We could have chosen version 1.7, but we think a major upgrade, with design and functionality changes, reflects a major version better.

The report and documentation are in English, not Dutch. English is the obvious choice as software development is often cross-border. Main reason: flexibility regarding the future developments of the product without much initial investments. Besides this, it is a good exercise for students.

Support for the current file format is continued. The application will display the test.xml file correctly as well as files that are in the wild. Customer trust and satisfaction are essential. If we dropped support then users will get stuck with presentations that can only be opened with older JabberPoint versions.

The file format is allowed to alter, but the implementation must be backwards compatible. This allows the introduction of new features while still supporting all presentations for all users. The alternative would be that the presentation file would not contain the theme. This could be done by not saving the theme at all or save it in another file that had to be related somehow with the presentation file. A good user experience and simplicity lead to the choice to alter the file format.

We made a choice to provide the themes with the presentation files. In this case there can be modifications to the theme and when transferring the presentation it is still possible to see the presentation as intended. We made the choice to be backward compatible. When there is no theme information in the presentation file the default theme will be used. In the application we build hard coded several themes. Normally we would have done this using an specific theme xml file which contains the different themes. However due to the time this would take to implement this we made the choice to just hard code these themes $theyarereachableusingtheView->Thememenu$. In the hard coded themes we only changed the font name. We didn't implement different colours in the hard coded themes. Using the presentation file you can set this information.

The menu item *Open* is implemented to let the user open a new presentation. This menu item did not provide this functionality. We assume that users appreciate this implementation. There were no specifications, but we believe the behaviour of the *Open* menu item is what users expect without much discussion. It can be extended with functionality, such as for example file filters, later on.

The menu item *New* is removed, because the user experience of just removing the presentation was not good. The no editing nature regarding presentations did not correspond to the creation of a new empty presentation. In the context of clear and unambiguous functionality, we decided to remove the menu item completely.

The menu item *Save* was kept, because it allows a user to save a theme selection. We decided that if a user has selected a theme for a presentation then the expected behaviour of the application that it would apply that theme to the presentation automatically. The user can store the theme by choosing the menu item. We did implement this partially. Internally we have a structure to be able to save the theme. However we did not completely implement all the

parts of the model we need to store. We however have an implementation of a test presentation which contains all the specifics we made. This functionality will be used when loading a file. As many textbooks write: saving the file is left as an excercise for the reader.

We decided to keep two switch statements because these are both clear and simple. There is a switch statement in a factory class, where occasionally switches or if statements are necessary to handle different parameter values. And there is a switch statement to handle the key that was pressed by the user. Different implementations like a strategy pattern or something else would result in more complex, thus less maintainable code. That this is a reasonable decision is apparent from the implementation of java itself, where the switch statement is used often in high profile classes such as `java.awt.event.KeyEvent`.

We decided to start with the implementation of localisation. For now it is kept simple, only supporting the en_US locale. With this, the application has centralised labels and is prepared for other languages in the future.

We designed the package structure according to the package-by-layer style. Generally, this style is easier to understand and more common than the package-by-feature style. If in the future a new view, like a web application, is required then we expect it to be easier to support as the view and controller classes are in well defined layers with well defined dependencies. Of course, the package-by-future style has advantages too, but we still preferred the layer style.

6 Sourcecode

6.1 Startup and load presentation

The entry point of the application is the `JabberPoint` class which is located in the main package. The main method starts with the creation of the two controller classes: `PresentationController` and `ApplicationController`. The view is created via the `SlideViewerFrame` class and the `ApplicationController` is registered as a listener to it. The view is then registered as a listener or observer to the `Presentation`. Now, the application setup is finished so is the `JabberPoint` class. The last thing to do is calling the method `loadPresentation` of the `ApplicationController` passing the optional command line parameter.

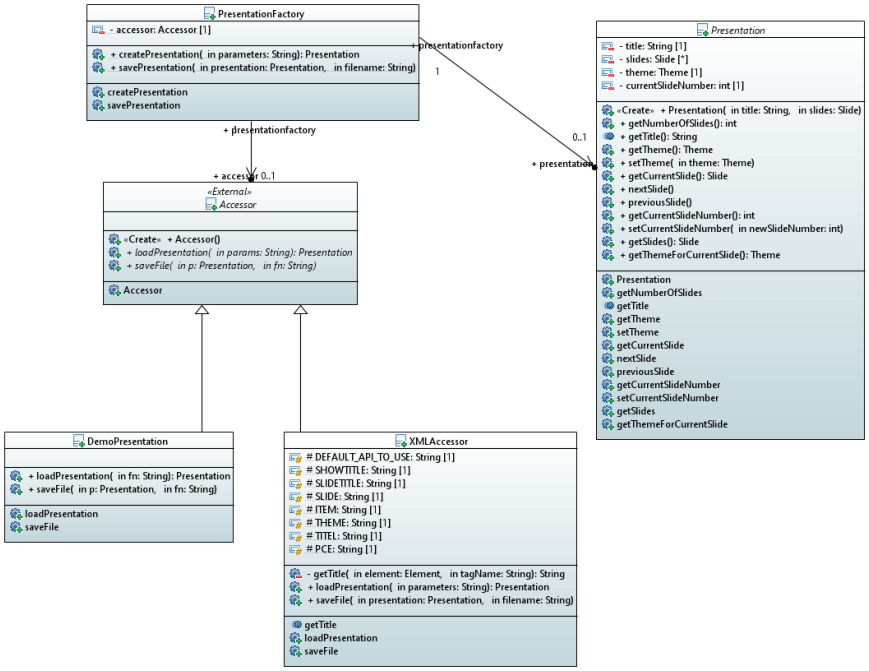
The `ApplicationController` the delegates the creation of a `Presentation` to the `PresentationController` and the `PresentationFactory`. The method `createPresentation` receives the optional command line parameter. The `PresentationFactory` instantiates an `Accessor` to load and return a `Presentation`. If no startup parameter was received then `DemoPresentation` is returning the `Presentation`, otherwise `XMLAccessor` will return a `Presentation` loaded from disk.

The `ApplicationController` tells the view, which implements the `PresentationObserver` interface to listen to the created `Presentation` object. Setting the current slide to 0 triggers the view to display the current `Slide` of the `Presentation`.

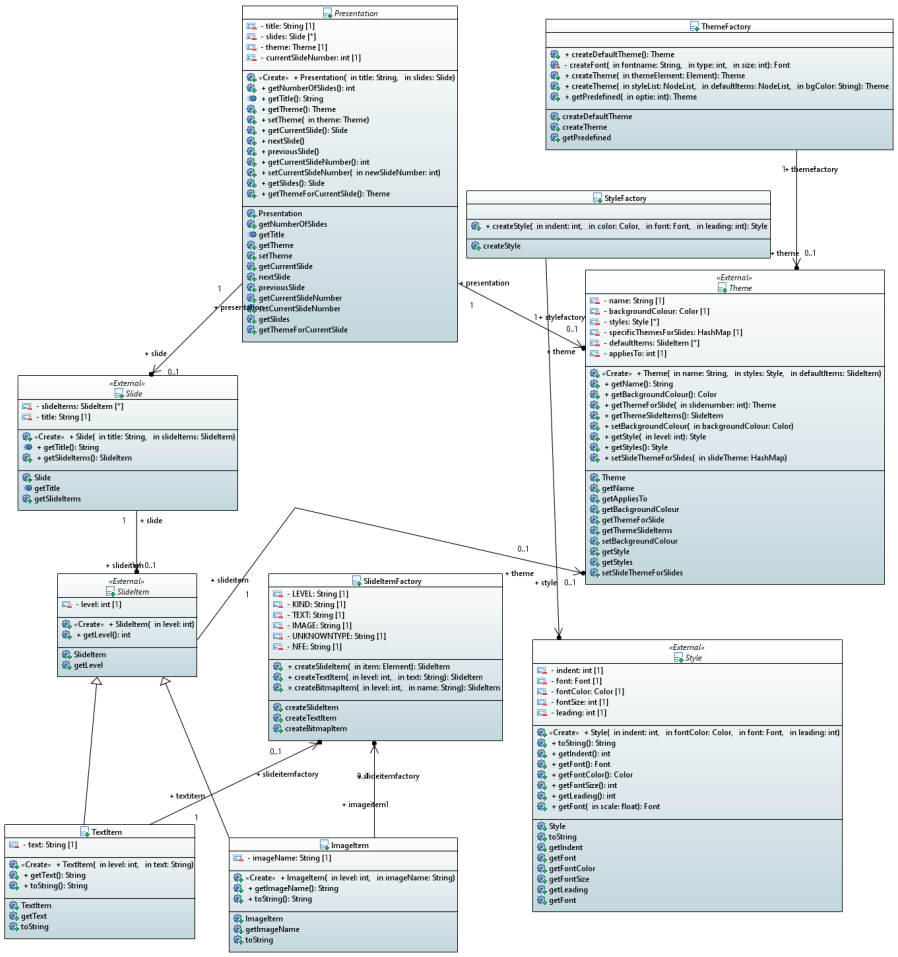
To facilitate testing the application comes with two XML files:

- `test.xml`: the original file delivered with the assignment
- `test-new.xml`: showcase of new functionality

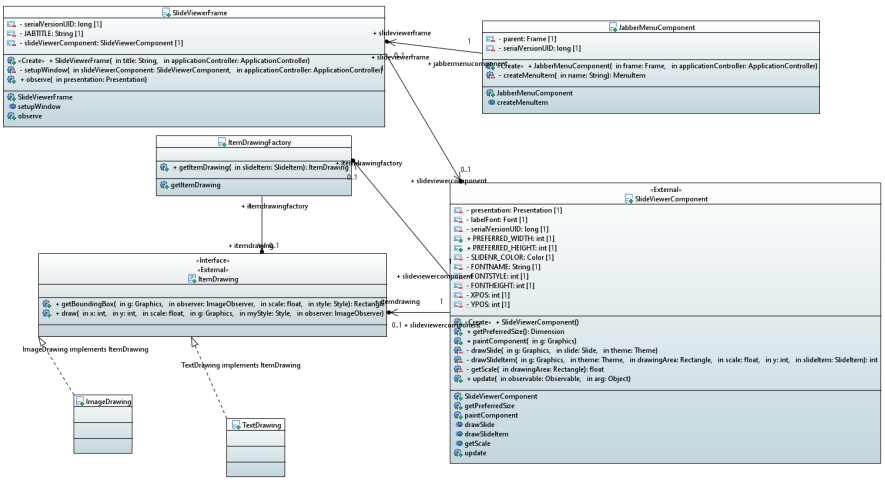
6.2 Domain classes



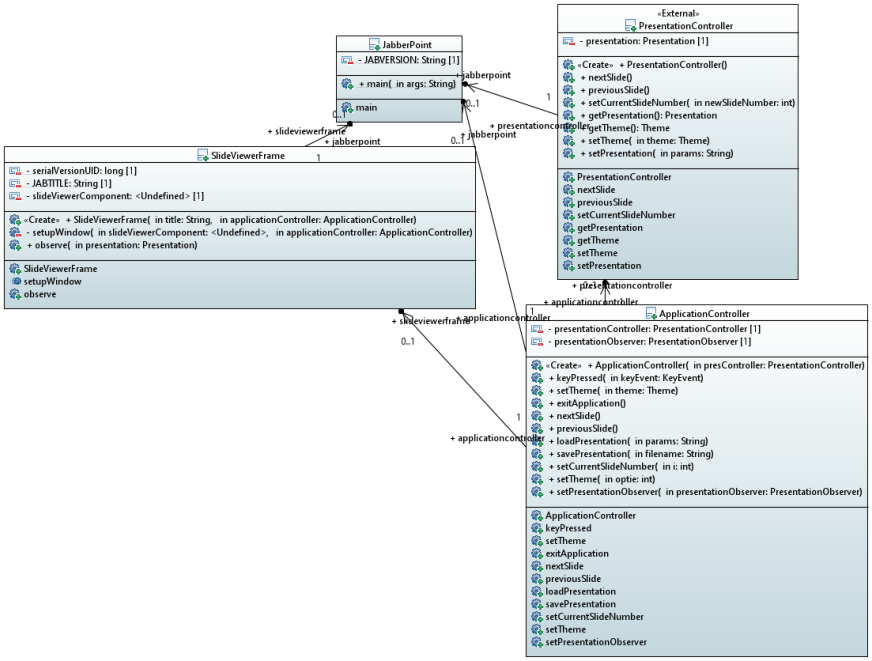
Figuur 2: IO package



Figuur 3: Model package



Figuur 4: View package



Figuur 5: Main package and dependencies