System design document for FabNotes

Version: 1

Date: 2017-05-27

Author: Group 7

This version overrides all previous versions.

1 Introduction

This application is meant for writing notes. The application is targeted towards students to make it easier to take notes. Its main purpose is to make note taking easier and more fun; everyone will be able to make the note their own style. It does not only have support for text, but also images and paintings. The idea is that if it's easier and more amusing to make a note, the user will also find it fun to learn.

1.1 Definitions, acronyms and abbreviation

- Textcontainer - Contains text, created when the user presses the note in "Write"-state.
- Imagecontainer - contains only images.
- Paintingcontainer - contains a canvas that the user may paint in while in "Paint"-state.

2 System design

Fabulous Notes, in short FabNotes, follows mainly some important design patterns that encapsulates logic in the easiest way possible in order to make the code easily read and easy to understand:

- Strategy Pattern [1]
  - Is used by the behavior the NoteObjects may have.
- State Pattern [2]
  - Is used by the fxml controllers to change the behavior of the user inputs on the note.
- Command Pattern [3]
  - Is used by the PaintingContainer to store and load paintings.

2.1 Overview

2.1.1 JavaFX

FabNotes's graphical user interface is developed using the JavaFX codebase. It supports all the elements wanted to build a desktop application gui and it also support designing elements by using Cascading Style Sheets (CSS).

2.1.2 States

Follows the state pattern [2]. The states implements the interface “NoteStateI” and are handled by the StateHandler in fxml/services. Each state contains the logic for what will happen when the user interacts with the note. When entering write state the user may click anywhere in the note to create a text area to write in, and when entering paint state the user may click anywhere in the note to create a canvas to paint in instead.

- StateHandler
  - Contains the current state and has the logic for changing states.
- WriteState
  - Is responsible for placing new text areas in the note.
- PaintState
  - Is responsible for placing new canvases in the note.

2.1.3 Behaviors

Follows the strategy pattern [1]. All the behaviors implements an interface “NoteObjectBehaviorI”. This makes it so that the client of a behavior doesn’t know anything about the logic behind the behavior. Thanks to this a behavior can be exchanged very easily during runtime without the client’s awareness.

- DragDropBehavior
  - Contains the logic used to move around note objects in a note, altering their graphical properties.
- ResizeBehavior
  - Contains the logic used mainly by ImageContainers to alter their properties and resize and rescale them.
- DragDropResizeBehavior
  - Marries the DragDrop and Resize Behaviors with each other. The ImageContainer makes use of this.
- PaintingBehavior
  -Contains the logic for painting in a canvas that is placed in the note.

2.1.4 Events

Everything that the user does to a note is event driven. If the user removes or adds something, or if the user moves an object, then the user will be able to redo these actions by simply pressing undo.

2.1.5 File management

The application at start creates (if it's not already there) a directory ".FabNotes" in the user's home directory. And in that directory a file is placed: "TAG_LIST.txt". That file holds all the available tags for the notes. All the notes will be saved in the same directory. The user is also, of course, able export notes and / or import notes from elsewhere, however only the notes in the FabNotes directory are shown in the file handler menu on the left in the application.

When a note is being saved, the class XMLWriter saves the note in XML format. See appendix.

2.2 Software Decomposition

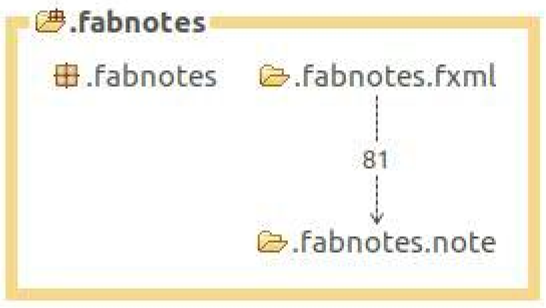


Fig. 1 Package diagram for FabNotes.

2.2.1 Decomposition into subsystems

The different top level packages: fxml and note.

The note package is completely independant. It has no dependencies outwards and may be used in another project by just placing it there.

The fxml package is also independent, however it has bridges that connects it to the note package. If one were to include this in another project, then the bridges and the states would need modifying to work properly. But it makes it easy since all the outside dependencies is placed in these two places, and not hard to change.

2.2.2 Package Descriptions

The desktop application consists of four top level packages and several underlying sub packages. Each package handles its own logic.

Top level packages:

- fxml:
  - Handles the logic for the .fxml files in resources. The .fxml files is the base for the gui.
  - Handles the logic for the application states.
  - Contains bridges for usage with outside resources.
- note:
  - Handles the logic for the current note the user is viewing.
  - Handles the logic for all the note objects that can be loaded or added to the note the user is building or importing.

2.3 Concurrency issues

Not applicable: *FabNotes* does not make use of any concurrent processing.

2.4 Persistent data management

Not applicable: *FabNotes* does not make use of any persistent data.

2.5 Access control and security

Not applicable: *FabNotes* does not make use of or require any internet access or any special system access. The local files that is used by the application are distributed by the application itself.

3 References

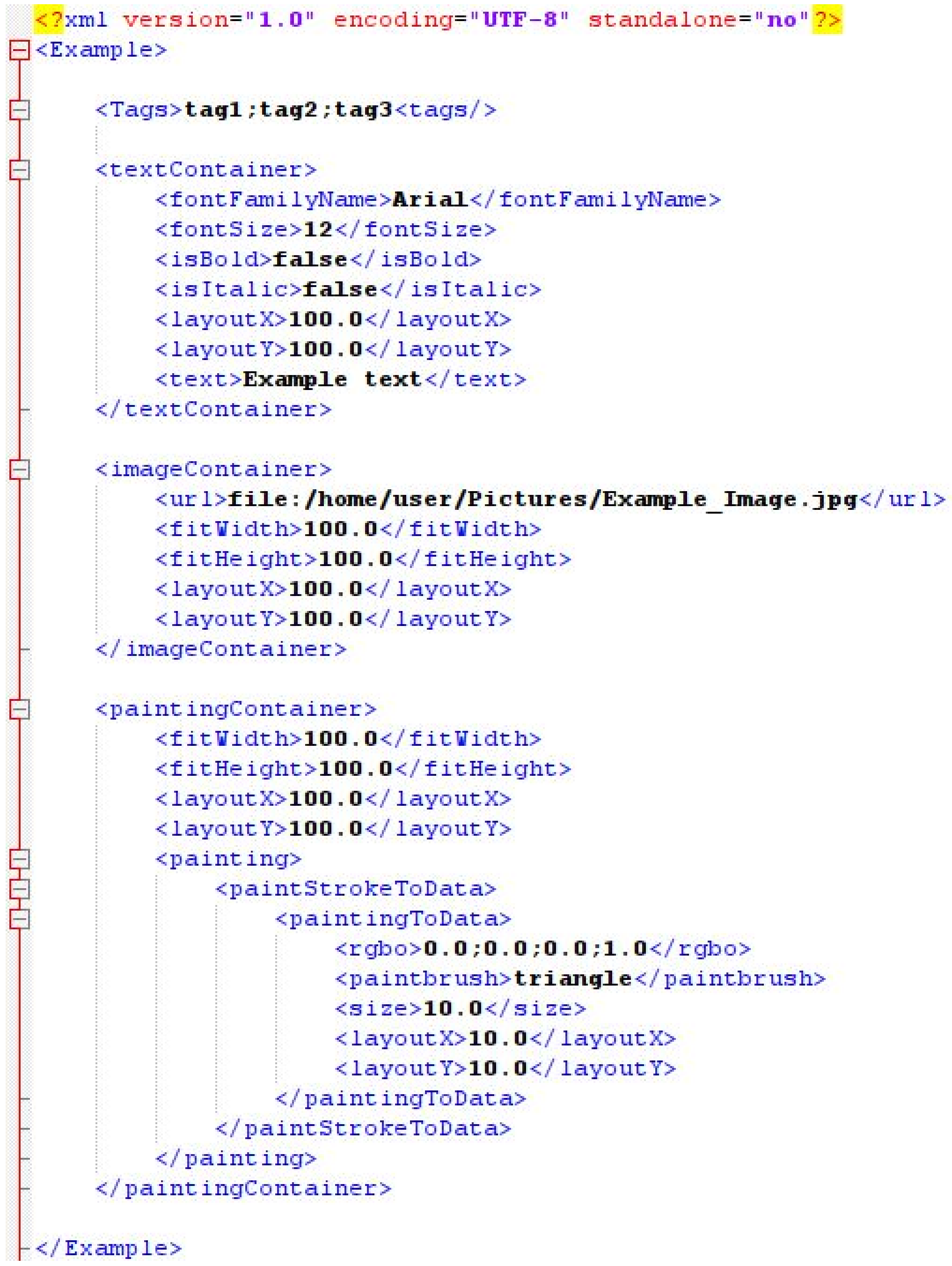
[1] Strategy Design Pattern. Tillgänglig:
https://sourcemaking.com/design_patterns/strategy, Hämtad: 2017-05-10

[2] State Design Pattern. Tillgänglig:
https://sourcemaking.com/design_patterns/state, Hämtad: 2017-05-10

[3] Command Design Pattern. Tillgänglig:
https://sourcemaking.com/design_patterns/command, Hämtad: 2017-05-21

Appendix

Example XML (a saved note)



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Example>

    <Tags>tag1;tag2;tag3<tags/>

    <textContainer>
        <fontFamilyName>Arial</fontFamilyName>
        <fontSize>12</fontSize>
        <isBold>false</isBold>
        <isItalic>false</isItalic>
        <layoutX>100.0</layoutX>
        <layoutY>100.0</layoutY>
        <text>Example text</text>
    </textContainer>

    <imageContainer>
        <url>file:/home/user/Pictures/Example_Image.jpg</url>
        <fitWidth>100.0</fitWidth>
        <fitHeight>100.0</fitHeight>
        <layoutX>100.0</layoutX>
        <layoutY>100.0</layoutY>
    </imageContainer>

    <paintingContainer>
        <fitWidth>100.0</fitWidth>
        <fitHeight>100.0</fitHeight>
        <layoutX>100.0</layoutX>
        <layoutY>100.0</layoutY>
        <painting>
            <paintStrokeToData>
                <paintingToData>
                    <rgbo>0.0;0.0;0.0;1.0</rgbo>
                    <paintbrush>triangle</paintbrush>
                    <size>10.0</size>
                    <layoutX>10.0</layoutX>
                    <layoutY>10.0</layoutY>
                </paintingToData>
            </paintStrokeToData>
        </painting>
    </paintingContainer>

</Example>
```

Fig. 2 Example, saved note

Packages

fxml

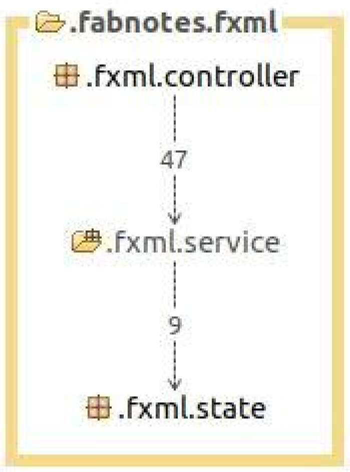


Fig. 3 fxml package

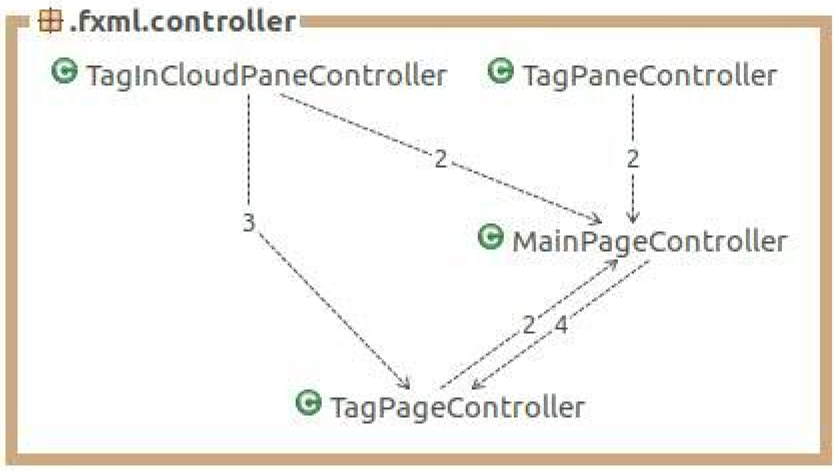


Fig. 4 fxml controller package

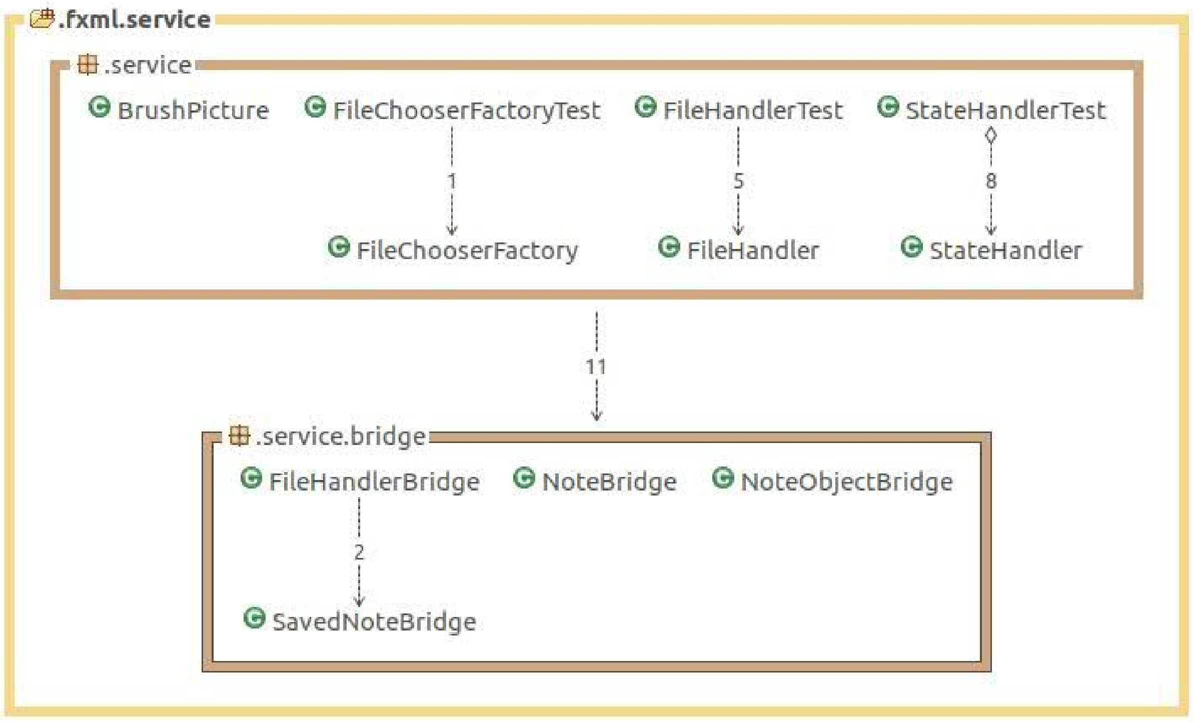


Fig. 5 fxml service package

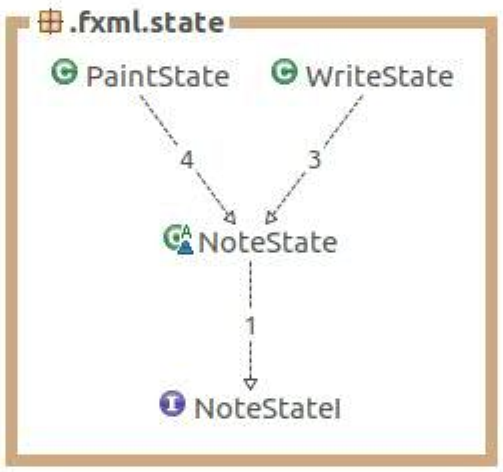


Fig. 6 fxml state package

note

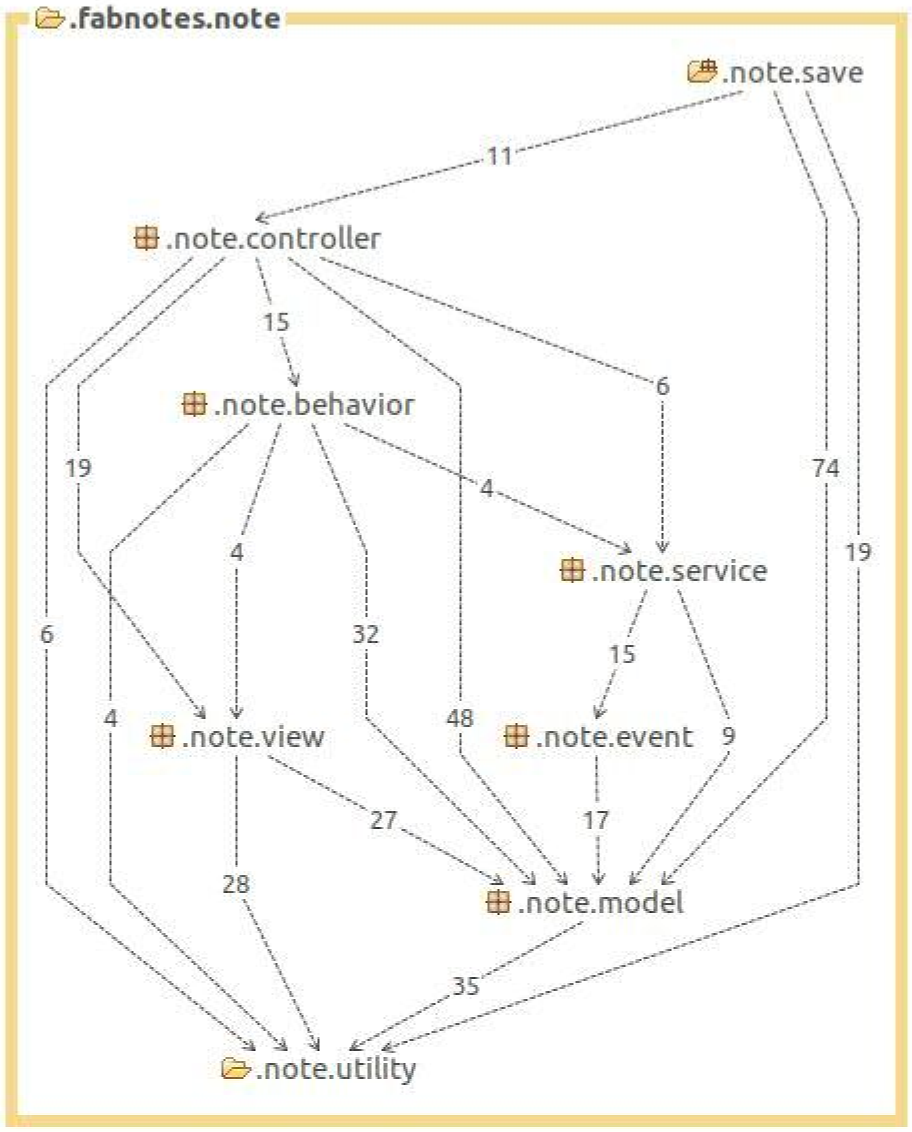


Fig. 7 note package

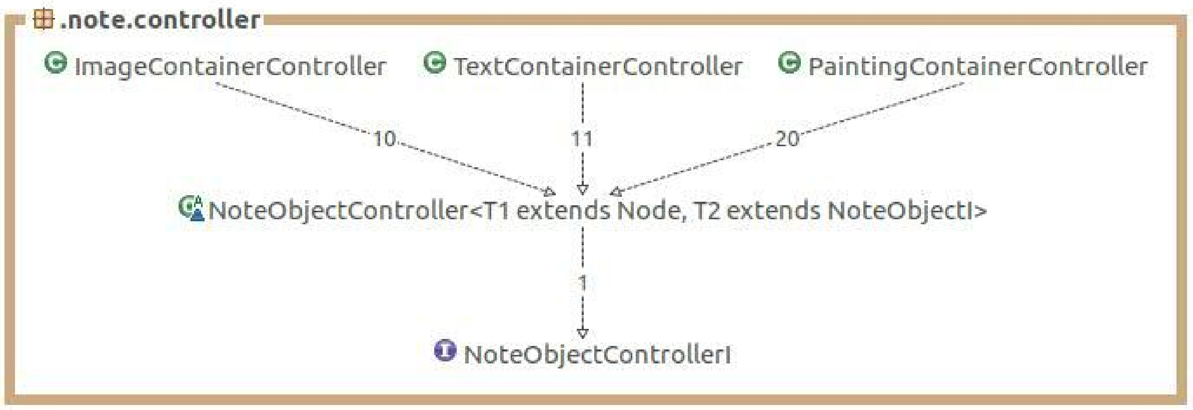


Fig. 8 note controller package

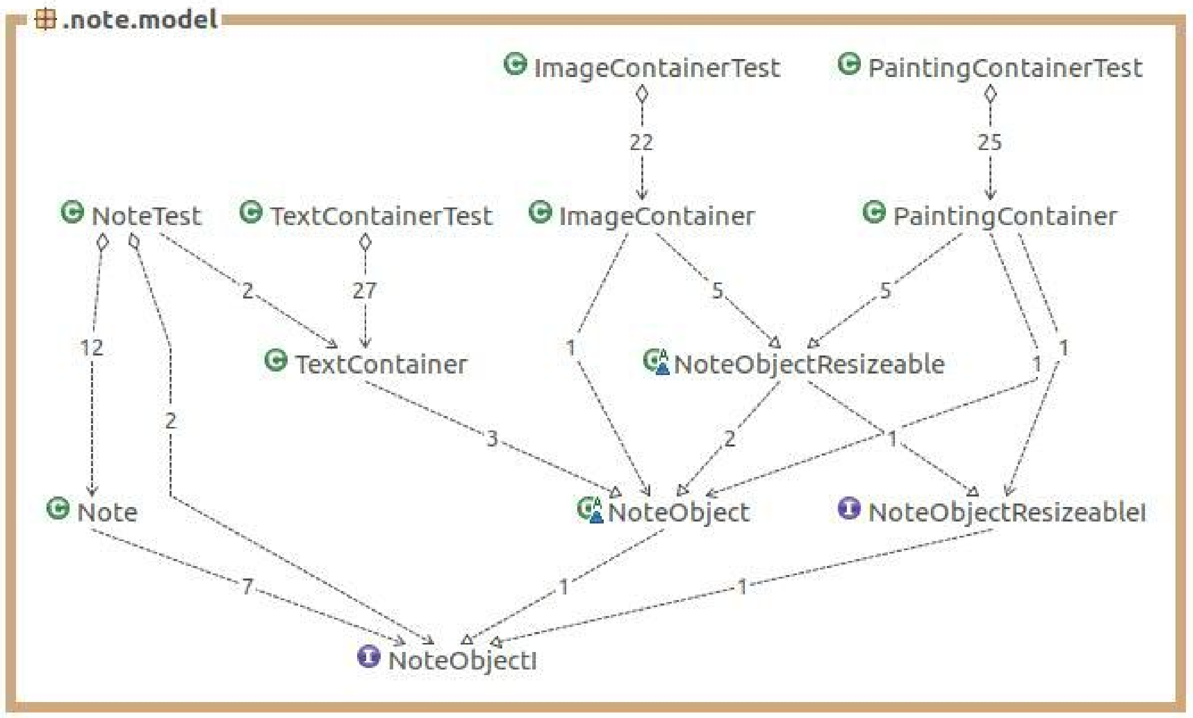


Fig. 9 note model package

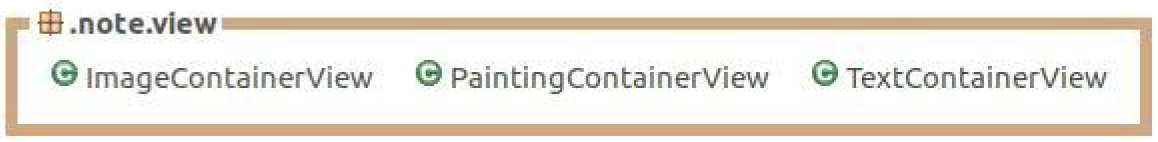


Fig. 10 note view package

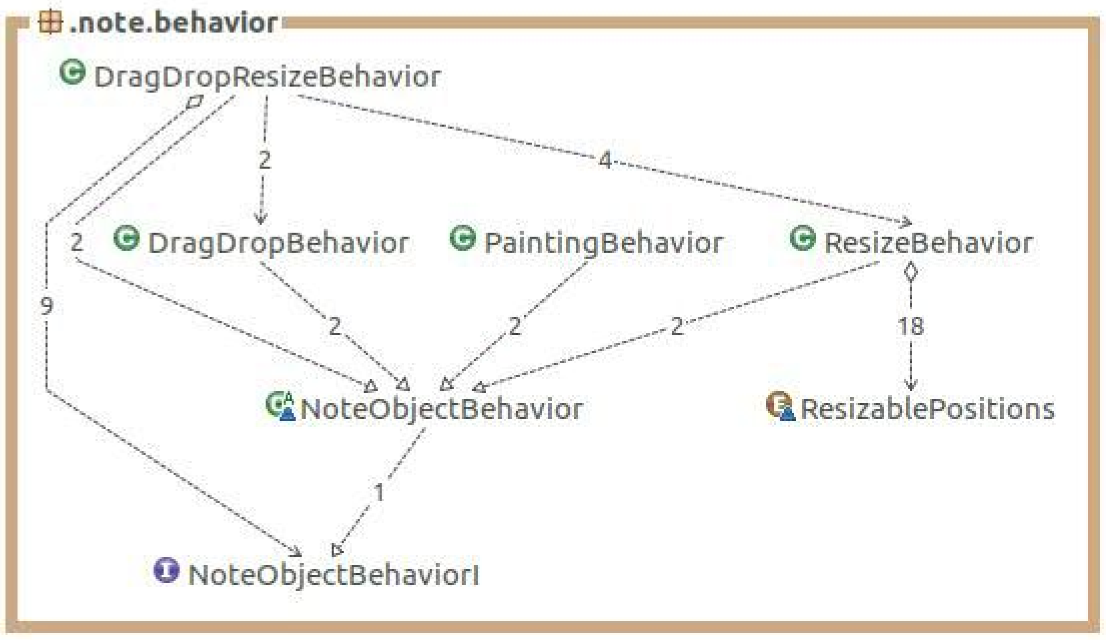


Fig. 11 note behavior package

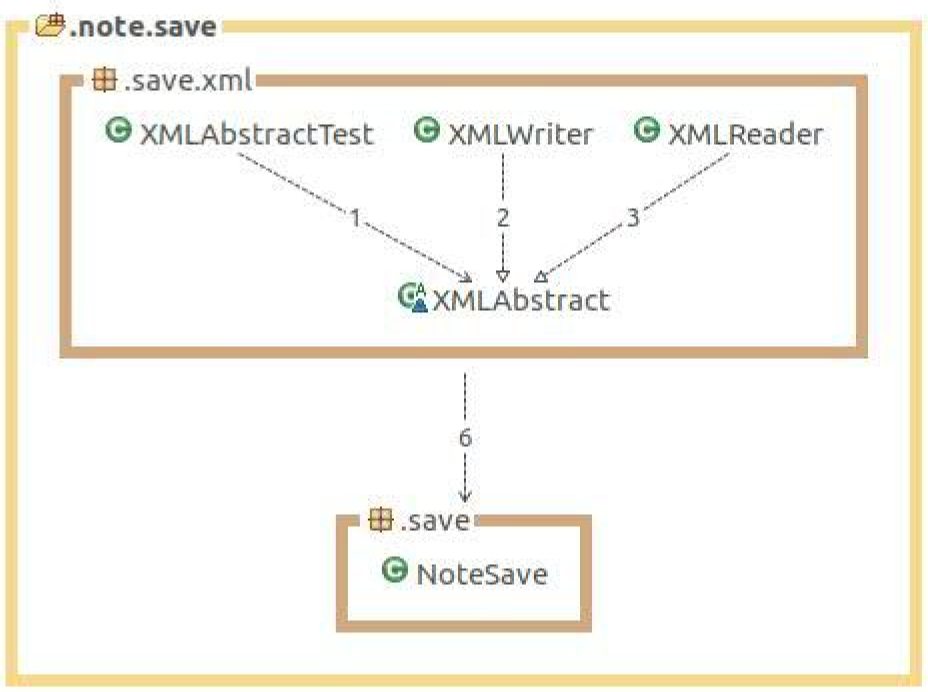


Fig. 12 note save package

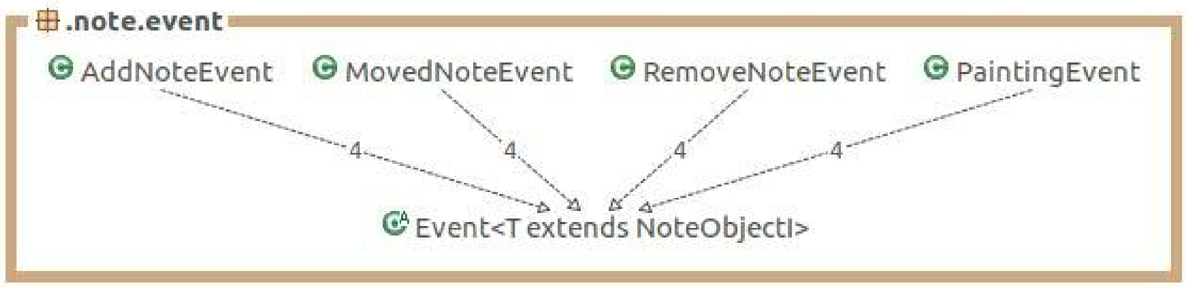


Fig. 13 note event package

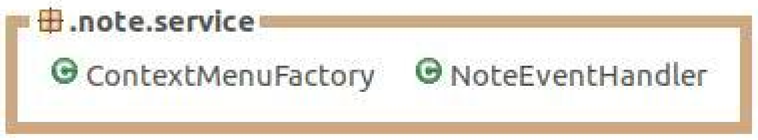


Fig. 14 note service package

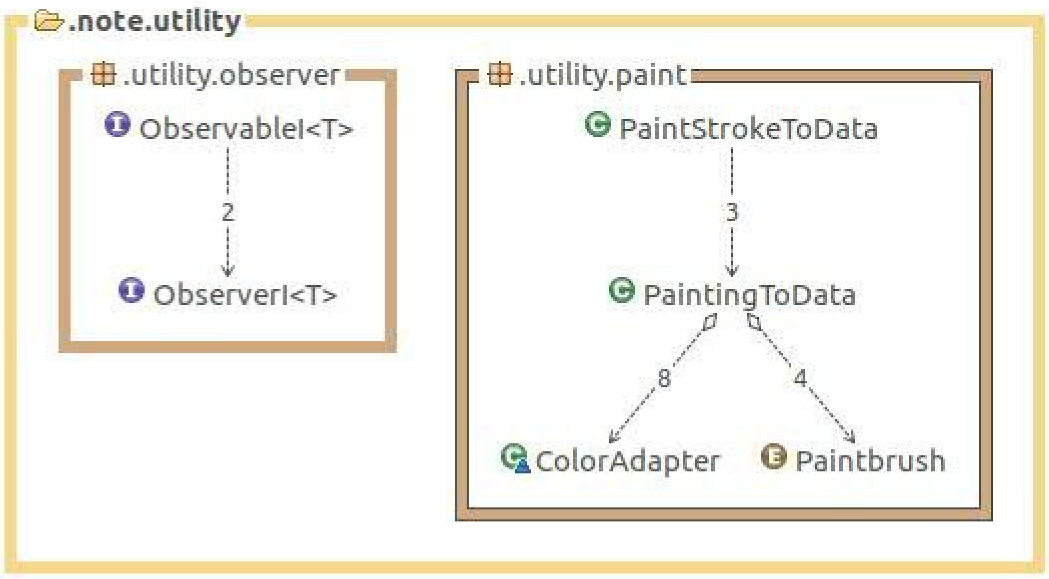


Fig.14 note utility package