Studentgegevens

Cursuscode IM0102

Opdracht Jabberpoint Inhoudsopgave

Naam Daniel S. C. Schiavini

Studentnummer 851102873

Aanpak

De oorspronkelijke bedoeling was om deze opdracht uit te voeren in een team met een andere student. Echter, er was geen andere student beschikbaar met een vergelijkbare tempo. Daardoor heb ik deze opdracht alleen uitgevoerd.

Voordat ik wist dat ik de opdracht alleen mag uitvoeren, had ik al een Git repository gemaakt. Ik heb de repository zo geconfigureerd dat dit document automatisch wordt gebouwd door het gebruik van TravisCI. Zodra een nieuwe versie van het rapport naar GitHub wordt gepushed, draait een script dat het PDF-bestand maakt. Het script bouwt ook Jabberpoint, draait unit tests, en als alles gelukt is wordt ook een JAR-bestand gemaakt. In de master branch hangt een deployment vast dat het PDF en JAR naar GitHub pages publiceert. Deze automatisering is iets minder nodig zonder team, maar het is nog steeds wel nuttig. De CI-configuratie is beschikbaar op github.com/DanielSchiavini-/design-patterns-assignment/blob/master/.travis.yml.

Jabberpoint had nog geen geautomatiseerde tests en omdat ik dat belangrijk vind, heb ik van tevoren JUnit tests geschreven (zie PR#5, PR#9 en PR#10). Echter niet alle klassen waren testbaar – TravisCI draait in een Linux-container waarin de Graphic Java-klassen niet gecreëerd kunnen worden. Ik had niet genoeg tijd had om dit probleem op te lossen, daarom heb ik alleen tests geschreven voor de domein- en accessor-klassen. Het probleem van UI-domein vermenging wordt verder besproken in sectie 3.4.

Het schrijven van de tests was een goede hulpmiddel om Jabberpoint te leren kennen en om het probleemanalyse van de volgende sectie beter uit te voeren.

1 Probleemanalyse

Om het probleem te analyseren kunnen we het beste de opdracht in eisen omschrijven:

- **Eis 1** : De gebruiker moet één of meer slides met inhoudsopgaven kunnen toevoegen aan een presentatie.
- **Eis 2** : Een inhoudsopgave-slide moet alle onderwerpen van de presentatie tonen.
- **Eis 3** : Het onderwerp moet slechts eenmaal worden getoond wanneer meerdere slides achter elkaar hetzelfde onderwerp hebben.
- **Eis 4** : De inhoudsopgave moet automatisch worden geüpdatet wanneer slides worden toegevoegd of verwijderd.

- **Eis 5** : De titel moet als onderwerp worden gebruikt wanneer een slide geen onderwerp heeft (**aanname**).
- **Eis 6** : Het onderwerp mag op slides worden getoond, behalve in de inhoudsopgave (optioneel).
- **Eis 7** : De onderwerpen in de inhoudsopgave mogen volgnummers krijgen (optioneel).
- **Eis 8** : Het onderwerp van de slide die na een inhoudsopgave komt, mag een speciale aanduiding krijgen (i.e. een alternatieve letterstijl) (optioneel).
- **Eis 9** : De stijl van de inhoudsopgave is vrij te bepalen.

De opdracht om een inhoudsopgave te implementeren in Jabberpoint is best interessant. Om een inhoudsopgave te tonen, is kennis nodig van de hele presentatie. Een verandering in elke slide kan invloed hebben op alle inhoudsopgaven. Deze circulaire afhankelijkheid maakt de opdracht uitdagend.

2 Ontwerp

Het onderwerp zoals geïmplementeerd verandert weinig van het originele Jabberpoint-ontwerp. Er is een nieuwe klasse toegevoegd voor de inhoudsopgaven, namelijk `TableOfContentsSlide`. Deze nieuwe klasse is een sub-klasse van `Slide`.

Klasse `TableOfContentsSlide` is uiteraard verantwoordelijk voor het genereren van de inhoudsopgave. Daarbij hebben we de optie om de slide-items die al geïmplementeerd zijn te hergebruiken. Het is logisch om de software zo optimaal mogelijk te hergebruiken, en niet alleen de slide-items maar ook het tekenen van deze items. De inhoudsopgave is in feite een slide die zijn eigen content genereert.

Het grootste verschil tussen de klassen is dat de inhoudsopgave kennis moet hebben van de hele presentatie. Daardoor krijgt een `TableOfContentsSlide` een referentie naar de presentatie in de constructor.

Zodra de inhoudsopgave getekend gaat worden, worden de presentatie-onderwerpen gelezen en wordt een nieuwe lijst van slide-items gemaakt. Daartoe kunnen wij de bestaande methoden van de `Slide` klasse gebruiken. Omdat er geen speciale stijl-eisen zijn voor de inhoudsopgave, kunnen wij eenvoudig de al ingebouwde levels gebruiken om de stijl te bepalen.

De aangepaste klassendiagram wordt in figuur 1 getoond.

Het voordeel van dit onderwerp is dat er vrij weinig hoeft te veranderen. De Accessor-klassen krijgen de extra verantwoordelijkheid om de inhoudsopgaven te maken en te schrijven. Voor de rest van de code is een inhoudsopgave een normale slide die zijn eigen content genereert. Accessor-klassen mogen zelf bepalen of er een referentie wordt opslagen naar een inhoudsopgave, of een lijst van de inhoud van de slides.

Voor de implementatie van de `draw` methode was een manier nodig om alle eerdere slide-items te verwijderen. Het is natuurlijk ook mogelijk om de `items` attribuut rechtstreeks te veranderen; maar het is in Java nooit verstandig om

Figuur 1: Klassendiagram voor de domein-objecten. Sterren geven de nieuwe klasse en methoden aan.

attributen van andere klassen te veranderen (zelfs van een super-klasse). Daartoe komt een nieuwe **protected** methode `clear` in de slide. De attributen van `Slide` zijn **private** geworden.

Het is verstandig om software te implementeren in de meest simpele manier, en dan itereren om deze te verbeteren. Daarom vind ik dat het bij de opdracht past om een aantal verbeteringen ook door te voeren. Dit wordt verder uitgelegd in sectie 3.

Een screenshot van de inhoudsopgave van de demo-presentatie wordt in figuur 2 getoond.

2.1 Traceerbaarheid van eisen

In deze sectie geef ik onderdelen van het ontwerp aan die verantwoordelijk zijn om de eisen (zie sectie 1) te realiseren. Dit geeft dus de relatie tussen probleemanalyse en ontwerp.

- De `Accessor`-klassen zijn verantwoordelijk voor **Eis 1**.

Figuur 2: Inhoudsopgave van de demo-presentatie in de master branch.

- De `TableOfContentsSlide` klasse is verantwoordelijk voor **Eis 2**, **Eis 3**, **Eis 4**, **Eis 5**, **Eis 7**, **Eis 8** en **Eis 9**.
- De `Slide` klasse is verantwoordelijk voor **Eis 6**.

3 Keuzen

In deze sectie gaan we een aantal vragen door die zijn opgeroepen tijdens het analyseproces.

3.1 Hoe moet de Slide klasse-structuur eruit zien?

Het is vanzelfsprekend dat `TableOfContentsSlide` een groot deel van de `Slide` code nodig heeft. Bij het analyseren van Jabberpoint heb ik gemerkt dat de nieuwe klasse eigenlijk *alle* code moet gebruiken. Het enige onderdeel dat niet nodig is voor de inhoudsopgave is het nieuwe onderwerp-veld.

Meestal als een variatie komt voor een bestaand concept, moet er een interface (of abstracte klasse) komen die het oude en het nieuwe concept implementeren. De rest van het systeem werkt dan met de interface en hoeft niets te weten van de exacte implementatie. Dit is hier ook het geval, de presentatie-klassen hoeven immers niet te weten of een slide een inhoudsopgave is of niet.

We zouden dus `Slide` abstract kunnen maken en een nieuwe klasse maken die naast de bestaande methoden een onderwerp heeft. Als we echter denken aan wat voor andere soorten `Slide` er in de toekomst nog komen, is het aannemelijk dat deze ook in de inhoudsopgave moeten worden getoond.

Een alternatief ontwerp is geïmplementeerd in PR #17. In deze PR heeft `Slide` geen items meer, maar alleen een titel en `draw`. Een extra klasse `Content-`

`Slide` is gemaakt die slide-items heeft. De `TableOfContentsSlide` maakt elke keer een nieuwe `ContentSlide`. Dit ontwerp is een geldig alternatief maar er staan niet veel voordelen tegenover de toegenomen complexiteit. De software wordt immers ingewikkelder en er moeten extra *ifs* en *casts* worden uitgevoerd (*code smells*).

Het alternatieve klassendiagram wordt in figuur 3 getoond.

Figuur 3: Alternatief klassendiagram voor de domein-objecten. Sterren geven de nieuwe klasse en methoden aan.

3.2 Wanneer moeten items worden gegenereerd?

De table of contents is in feite een slide met een inhoud die gegenereerd wordt. De vraag is wanneer deze generatie moet gebeuren. In het gegeven ontwerp wordt de slide opnieuw gegenereerd in elke `draw`.

Een alternatief zou zijn om een manier te implementeren om de `TableOfContentsSlide` te laten weten als iets verandert. Bijvoorbeeld het observable patroon zou hier gebruikt kunnen worden.

Dat is echter niet nodig, de generatie is redelijk efficiënt, immers `draw` wordt alleen aangeroepen als de slide zichtbaar komt. `SlideViewerComponent` roept

namelijk `repaint` aan wanneer de slide op het scherm komt.

3.3 Welke klasse is verantwoordelijk voor onderwerpen?

De inhoudsopgave moet toegang krijgen tot een lijst van onderwerpen in de presentatie. In het ontwerp krijgt `TableOfContentsSlide` een referentie naar de presentatie en is zelf verantwoordelijk om deze te lezen. En alternatief hierop is om de presentatie verantwoordelijk te houden voor de onderwerpen in de slides. `TableOfContentsSlide` zou dan een dan een onderwerpen-lijst ontvangen.

De voordelen zijn:

- De inhoudsopgave hoeft niets te weten van de presentaties.
- De presentatie is verantwoordelijk om slides te lezen.
- De circulaire referentie wordt opgeheven.
- De lijst van onderwerpen kan tussen de verschillende inhoudsopgaven worden gedeeld.

De nadelen zijn:

- Er komt meer logica in de presentatie die alleen nodig is voor de inhoudsopgave.
- De inhoudsopgaven moeten nog steeds alle items doorlopen om te bepalen welk onderwerp het volgende is.
- De methoden die nodig zijn om de slides te lezen bestaan al in de presentatie.
- Het is conceptueel correct om een circulaire referentie te hebben, immers een inhoudsopgave moet een overzicht van de presentatie laten zien.
- De presentatie klasse wordt nog complexer.

De keuze om de onderwerpenlijst in de inhoudsopgave te implementeren geeft dus een betere inkapseling.

3.4 Moeten UI- en domeinobjecten worden gesplitst?

Het feite dat slides en slide-items verantwoordelijk zijn om zichzelf te tekenen heeft wel voordelen. Dit volgt het principe van inkapseling, immers alleen de klassen zelf weten hoe ze eruit moeten zien.

Echter het testen en uitbreiden van de software wordt heel lastig. Zodra gegevens gekoppeld worden aan presentatie kunnen we moeilijk de gegevens ergens anders vandaan halen. Integreren van een database of een API wordt dan lastig. Ook het testen van de applicatie is moeilijker (zie sectie Aanpak). Bovendien kunnen we de gegevens moeilijk in andere omgevingen laten zien.

Deze fout komt vaak voor, zeker in grote projecten. De aanname dat het een desktop-applicatie betreft wordt verspreid door de code, testen en uitbreiden kost steeds meer moeite. Ik zou graag de gegevens willen scheiden van AWT-afhankelijkheden, maar dit is geen kleine opgave en valt buiten de scope van deze opdracht.

In een persoonlijke notitie, mijn huidige klant loopt tegen hetzelfde probleem aan in een veel grotere schaal. Wij willen de logica van hun applicatie met 500.000 gebruikers nu ook in de cloud draaien, maar zonder de gebruikersinterface-afhankelijkheden gaat dit niet eenvoudig.

3.5 Moet een inhoudsopgave verplicht een titel hebben?

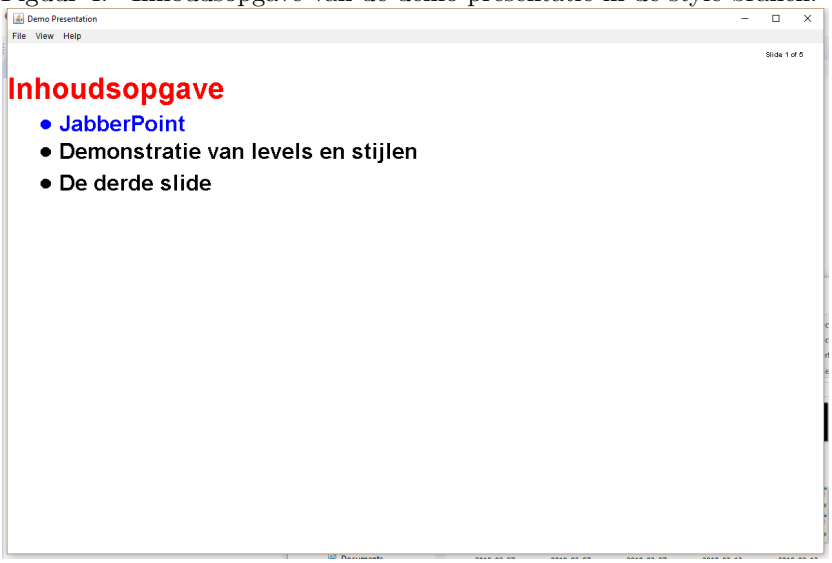
In de opdracht wordt niet benoemd of elke inhoudsopgave verplicht een titel moet krijgen. De opgeleverde versie van de applicatie geeft een standaard-titel ("inhoudsopgave") als er geen titel wordt gegeven. Echter aangezien alle slides een titel nodig hebben, is het niet raar om de titel van de inhoudsopgave ook verplicht te maken. Deze aanpassing is geïmplementeerd in PR#21.

3.6 Moet een inhoudsopgave een andere stijl krijgen?

De standaard stijl van de inhoudsopgave is niet optimaal, alhoewel **Eis 9** expliciet aangeeft dat stijlen vrij te bepalen zijn. Het is echter niet helemaal duidelijk dat het om een lijst van onderwerpen gaat.

Door een icoontje toe te voegen aan elke item en steeds dezelfde indentatie te gebruiken, kan dit verbeterd worden. Deze verbetering is doorgevoerd in PR#22. Een screenshot van de inhoudsopgave met deze stijl wordt in figuur 4 getoond.

Figuur 4: Inhoudsopgave van de demo-presentatie in de style branch.



Anderzijds kunnen we volgnummers toevoegen aan elke item om hetzelfde probleem op te lossen. Deze oplossing wordt benoemd in **Eis 7** en geïmplementeerd in PR#23. Een screenshot van de inhoudsopgave met deze stijl wordt in figuur 5 getoond.

removed versions javadoc lambdas in controllers

Figuur 5: Inhoudsopgave van de demo-presentatie in de numbers branch.

4 Sourcecode

In deze sectie geef ik links naar de verschillende resultaten van de opdracht. Alle links betreffen zowel Jabberpoint als dit rapport.

- Jabberpoint code: github.com/DanielSchiavini/design-patterns-assignment-/tree/master/Jabberpoint
  De werking ervan wordt uitgelegd in sectie 2.
- LaTeX code: github.com/DanielSchiavini/design-patterns-assignment/tree-/master/Report
- Aanpassingen op Jabberpoint: github.com/DanielSchiavini/design-patterns-assignment/compare/v0.0.1...master.
- Gegenereerde bestanden: github.com/DanielSchiavini/design-patterns-assignment/tree/gh-pages
- Laatste build-rapport van TravisCI: travis-ci.com/DanielSchiavini/design-patterns-assignment/
- Rapporten van TravisCI: travis-ci.com/DanielSchiavini/design-patterns-assignment/builds
- Optionele implementaties: github.com/DanielSchiavini/design-patterns--assignment/pulls