

**Object Georiënteerd Ontwerp**

**Zelfevaluatie App**

Steven Zegers

Wout De Boeck

Thibault Stroobants

2018 Toegepaste Informatica Reeks 2

23/12/18

University Colleges Leuven-Limburg  
Campus Proximus  
Geldenaaksebaan 335  
3001 Heverlee  
Tel. +32 (0) 16 375 700

Inhoud

[Algemene opmerking 3](#_Toc499494737)

[Vereisten 3](#_Toc499494738)

[Design patterns 5](#_Toc499494739)

[Speciale topics 7](#_Toc499494740)

[Werkverdeling 8](#_Toc499494741)

[Klassen diagram 9](#_Toc499494742)

# Algemene opmerking

Dit verslag gaat over de opdracht ‘Zelfevaluatie app’, dat zal meetellen voor 5 punten van je totaalscore van dit vak. Dit verslag vormt de toelichting bij jouw code.

**Je wordt geacht om je voor het verslag volledig aan deze template te houden. Buiten deze ‘algemene opmerking’ zijn alle paragrafen verplichte onderdelen van het verslag. Je print dit verslag af (dubbelzijdig) – 1 exemplaar per groep en levert dit in bij het einde van het laatste labo OOO.**

**Je maakt een zip file van al je source code (.java bestanden, geen .class bestanden) en van alle bestanden die van belang zijn voor deze OOO opdracht. Je voegt tevens de laatste versie van je verslag (Word document) toe aan deze zip file.**

**Je maakt tevens een jar file van je programma die kan runnen onder Java 8 (of hoger)**

**. Je laadt deze zip file en jar file op via Toledo uiterlijk voor maandag 24/12 23.59 uur. Naam van de zip file: Zelfevaluatie\_ gevolgd door de familienamen van de groepsleden gescheiden door underscore (bv. Zelfevaluatie\_Dox\_Lenaerts\_Jansen.zip). Idem voor jar file (bv. Zelfevaluatie\_Dox\_Lenaerts\_Jansen.jar)**

**Het spreekt voor zich dat de code op Toledo overeenstemt met de code in je repository op GitHub.**

# URL GITHUB repository

**Copy/paste hier de URL van je Github repository met je zelfevaluatie app project**

# URL

https://github.com/steven-zegers/OOO\_projects

# Vereisten

Geef voor het gehele project aan welke vereisten (evt. zelf verder uitgewerkt/opgesplitst aan de hand van de opgave) je succesvol hebben geïmplementeerd, en welke topics niet gelukt zijn. Indien je een deel van de voorziene vereisten niet afgewerkt hebt, geef dan aan waarom niet… De reden kan “tijdsgebrek” zijn, het kan een issue zijn “wist niet hoe, het crashte”, of het kan zijn dat je een zeer goede reden had om het niet te implementeren…

Voeg het definitieve gegenereerde klassendiagramma van je code toe, als afzonderlijke afbeelding, als bijlage bij dit rapport.

Als er zaken zijn uit de opdracht die je niet hebt kunnen uitwerken of die je beter zou willen uitwerken, dan som je deze hier tevens op (in laatste rij van tabel).

|  |  |  |
| --- | --- | --- |
| User story | OK? | indien niet ok – wat lukt niet (zie acceptatie-criteria) en waarom? |
| 1. Overzicht categorieën tonen | OK |  |
| 1. Nieuwe categorie toevoegen | OK |  |
| 1. Overzicht vragen tonen | OK |  |
| 1. Nieuwe vraag toevoegen | OK | (als duiding, om een statement dat reeds is toegevoegd in het nieuwe vraag venster te verwijderen moet het volledige statement in het statement veld ingevuld worden en dan op verwijderen gedrukt worden, wij dachten dat dit de bedoeling was ook al is dit niet super handig) |
| 1. Test afleggen | OK |  |
| 1. Resultaat tonen met punten | OK |  |
| 1. Resultaat tonen met feedback | OK |  |
| 1. Type feedback kiezen | OK |  |
| 1. Resultaat tonen zonder fouten | OK |  |
| 1. Test reeds afgelegd | OK |  |
| 1. Test opnieuw afleggen | OK |  |
| 1. Bestaande categorie aanpassen | OK |  |
| 1. Bestaande vraag aanpassen | OK |  |
| 1. Test selecteren uit lijst van excel bestanden | NOK | Geen tijd meer om dit te implementeren ☹ |

# Design patterns

Geef voor elk gezien patroon aan waar je het gebruikt hebt (mogelijkerwijs meer dan eens). Genereer een klassendiagram voor elke situatie waarin je het patroon gebruikt hebt.

Geef extra informatie (voordelen / waarom / …). Als je een patroon niet toegepast hebt, leg uit waarom niet.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Toegepast (ja/nee)  In welke stories(nr)  Waarom toegepast(voordeel) | | Bijhorend klassendiagram (gegenereerd vanuit je java code) |
| Observer | Ja, we zijn dit beginnen gebruiken bij story 2 en 4 aangezien we bij het toevoegen van nieuwe categorieën en questions ook wilden dat onze overview panes werden geüpdatet. De verschillende panes die worden geüpdatet implementeren de observer interface.  Daarna hebben we dit patroon in meerdere andere stories verder gebruikt door waar nodig nieuwe observers te maken en die te laten luisteren naar de subject (bij ons de facade klasse). Een voordeel hiervan is ook het “loose coupling”, façade weet niet concreet hoe de verschillende observers eruit zien maar weet wel dat deze allemaal een update methode hebben aangezien ze de Observer interface implementeren. | |  |
| Strategy | Ja, in de database klasses, aangezien het uitlezen van de files afhangt van of het gaat om tekstfiles, of dat het gaat om een tekst excel bestand bij het uitlezen. Er zijn dus verschillende implementaties van databases, moest er een nieuw soort database worden toegevoegd dan zou dit heel gemakkelijk gaan. Het strategy design patroon sluit aan bij het Open/closed principle, een nieuwe soort database zal enkel werk vereisen aan de nieuwe strategy die wordt toegevoegd, aangezien de concrete implementatie verborgen zit achter een reeds bestaande interface zal er aan de context klasse zo goed als niks moeten veranderd worden.  Het strategy patroon zou daarom ook kunnen toegepast worden bij de verschillende types van feedback met dezelfde redenering. Momenteel is dit nog niet het geval. | |  |
| Simple Factory | Ja, ook dit hebben we gebruikt voor de database klassen. Aangezien er verschillende soorten databases zijn kunnen we deze samenbrengen in een factory die dan afhankelijk van de soort die wordt aangegeven in de facade klasse zal worden aangemaakt. De factory heeft als enige responsabiliteit het creëren van objecten. | |  |
| Façade | Ja, deze klasse is eigenlijk een aanspreekpunt voor de controllers en UI. Ze hoeven de klassen en hun implementaties die achter de façade liggen niet te kennen. Hierdoor wordt het gebruik van de achterliggende klassen gemakkelijker. De facade verbergt dus de complexiteit. | |  |
| Singleton | Ja, ook dit hebben we gebruikt bij het implementeren van onze database klassen om er zeker van te zijn dat er altijd maar een en dezelfde instantie bestaat van het object. Moesten er toevallig twee verschillende instanties van onze database objecten zijn die op verschillende plaatsen gebruikt worden zou het kunnen dat de data niet volledig overeenkomt. |  | |
| State | Nee, dit zou op zich wel kunnen gebruikt worden om de verschillende stadia van een test te kunnen bijhouden (nog niet gestart, beëindigd, aan het oplossen, …). Echter hadden wij hier aan het begin van de opdracht niet aan gedacht en wegens tijdsnood hebben we dit niet meer geimplementeerd. | |  |
| MVC | Zo ver dit mogelijk is hebben we het MVC patroon toegepast, echter kennen de views wel de facade, dit zou kunnen weggewerkt worden maar lijkt ons in dit geval niet efficiënt | |  |
| Decorator | Nee, ons inziens is het gebruik van het decorator pattern niet van toepassing | |  |
| Template method | Ja, aangezien we zagen bij het omzetten van de textfiles er nogal wat dubbele code was hebben we dit gerefactored op een manier waarbij we template gebruiken. De abstracte klasse TextDatabase bevat daarin de stukken code die overeenkwamen in de 2 subklassen, de code die voor beide subklassen specifiek zijn staan dan als abstracte functie in de TextDatabase klasse en worden pas geïmplementeerd in de subklassen. Het uitlezen van de databases beschouwen wij als een algoritme. Daarna heeft dit design patroon ons ook nog veel tijd bespaard toen we onze code moesten aanpassen zodat deze ook werkte met een jar file vanop de command line. We moesten namelijk enkel code aanpassen in de gemeenschappelijke superklasse aangezien het uitlezen van een tekstfile toch gemeenschappelijke code was. | |  |

# Speciale topics

Geef voor elke “speciale topic” aan of je het gebruikt hebt of niet, en zo ja, waar. Toon eventueel aan met een klassendiagram.

|  |  |  |
| --- | --- | --- |
|  | Toegepast (ja/nee)  In welke stories(nr)  Waarom toegepast(voordeel) | Bijhorend klassendiagram (uit object aid) of extra uitleg |
| Enum | Ja, voor de database factory zodat we gemakkelijk kunnen bijhouden welk soort database types er zijn. Een nieuwe soort database toevoegen aan onze factory wordt hierdoor heel makkelijk. |  |
| Properties | Ja, stories 8 en 10, de properties file houdt bij welke instellingen de gebruiker graag zou gebruiken. Op basis hiervan worden dan bepaalde zaken geïnitialiseerd. |  |
| Reflection | Nee, wegens moeilijkheden bij het gebruik van reflection in combinatie met Singleton hebben we dit achterwege gelaten. |  |
| Andere… |  |  |

# Werkverdeling

Geef aan in percentages hoeveel je bij benadering gespendeerd hebt aan deze opdracht.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Steven Zegers | Wout De Boeck | Thibault Stroobants | Totaal |
| Ontwerp | 33% | 33% | 33% | 100% |
| Klassendiagrammen | 33% | 33% | 33% | 100% |
| Implementatie | 45% | 10% | 45% | 100% |
| Verslag | 60% | 0% | 40% | 100% |

# Klassen diagramma’s

Voeg het definitieve gegenereerde klassendiagram van je code toe), als afzonderlijk afbeelding-bestand. **Dit klassendiagram moet goed leesbaar zijn.** Spreid het over meerdere pagina’s (bv. 1 blad per package (MVC) en 1 overzicht klassendiagram (zonder attributen en methoden).







