

# OOP2 – Practicumopdracht

## Week 4

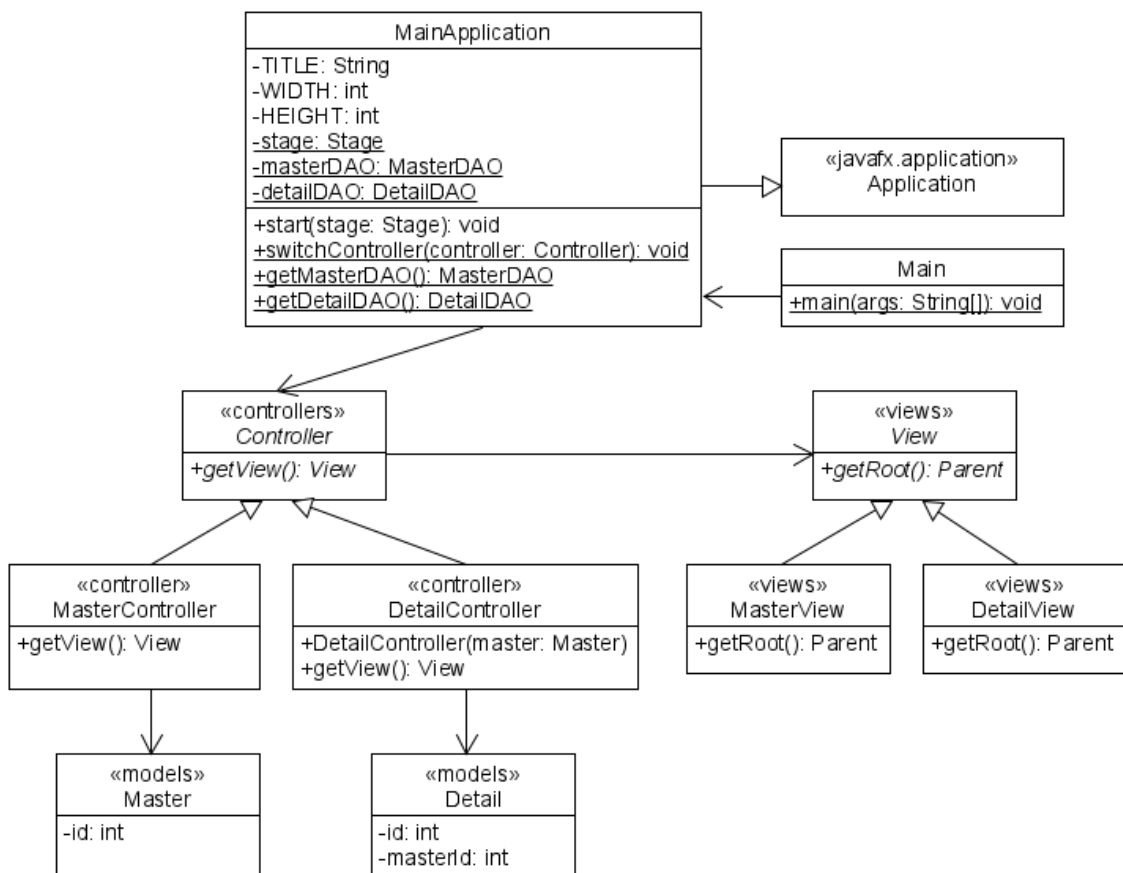
Nu je applicatie alle onderdelen van MVC bevat, wordt de focus verschoven naar een ander belangrijk onderdeel: de dataverwerking.

Je gaat de mogelijkheid creëren om onder andere de ListViews en ComboBox daadwerkelijk data te laten tonen. Om dit voor elkaar te krijgen gaan we gebruik maken van het *Data Access Object Design Pattern* (DAO).

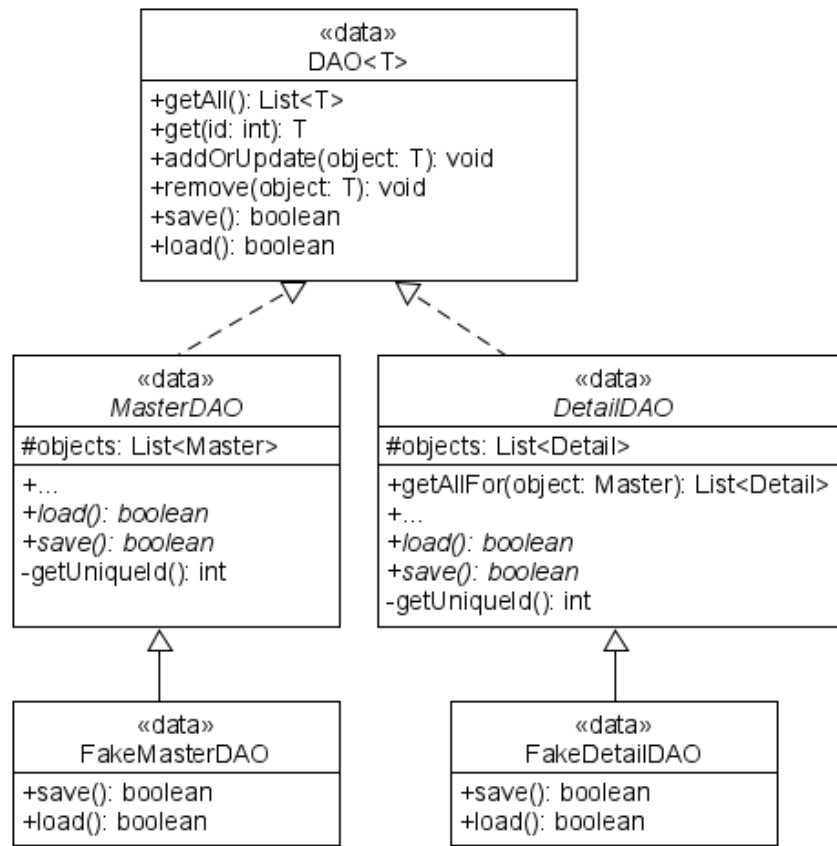
Een DAO in de context van een applicatie heeft eigenlijk één taak: Ervoor zorgen dat je met een databron kan communiceren alsof het een Array is, zonder dat alle overige lagen in de applicatie hoeven te weten wat voor databron dat precies is.

Bij een goede implementatie van dit Design Pattern zal het de applicatie dus niks uitmaken of je de databron een tekstbestand of een MySQL database is. Daarbij is vervangen van een DAO straks een kwestie van één regel code veranderen, wat voor het onderhouden van code wel zo fijn is.

Gelukkig houden we het voor deze week nog even simpel en gaan we data nog niet daadwerkelijk opslaan en inladen, maar gaan we wel de basis leggen om dit in de nog komende weken gemakkelijk te kunnen doen. Schrik niet van de grote stappen die volgen in het stappenplan, er is voor deze week gewoon iets meer begeleiding nodig om je in de goede richting te drukken voor de nog komende weken.



Afbeelding: UML met de gewenste aanpassingen aan de structuur (deel 1).



Afbeelding: UML met de gewenste nieuwe classes en interface (deel 2).

Werk nu het volgende stappenplan af met gebruik van de bovenstaande twee UML diagrammen:

1. Maak een package met de naam *data* in de *practicumopdracht*-package. Maak een nieuwe interface voor de DAO uit bovenstaande UML.
2. Breid je bestaande Master- en Detail-model uit zoals aangegeven in bovenstaande UML.
3. Maak de classes voor de MasterDAO en DetailDAO zoals aangegeven in de UML. Gebruik overal in plaats van de termen "Master" en "Detail" de namen van de models die je in de 1<sup>e</sup> week van de practicumopdracht hebt bedacht, bijvoorbeeld: *TodoLijstDAO* en *TodoRegelDAO*. Dit geldt ook voor de nog komende stappen.

4. Breid je MasterDAO zodanig uit, dat deze de volgende werking heeft:

- Zorg ervoor dat bij het instantiëren van de MasterDAO, direct de *load*-methode wordt aangeroepen.
- De *getAll*-methode returned alle objecten uit *objects*-List. Zorg ervoor dat je niet direct de *objects*-List zelf returned, maar een kopie met daarin dezelfde objecten. Anders kan de applicatie de originele List van buitenaf beïnvloeden (bijvoorbeeld nieuwe objecten toevoegen), terwijl dat de taak van de DAO is en moet zijn.
- De *get*-methode zoekt een object in de *objects*-List op basis van de gegeven *id* en returned deze.
- De *addOrUpdate*-methode controleert of:
  - Het binnenkomende object een *id* heeft die groter is dan 0. Als dat het geval is, wordt een bestaand object met dezelfde *id* uit de *objects*-List verwijderd en vervangen met het nieuwe object. Zorg ervoor dat bij het vervangen het nieuwe object op dezelfde index in de *objects*-List terecht komt.
  - Indien het binnenkomende object een *id* heeft die lager of gelijk is aan 0, dan krijgt het object een unieke *id*-waarde toegewezen en wordt het object aan de *objects*-List toegevoegd.
- De *remove*-methode verwijderd het binnenkomende object uit de *objects*-List. Zorg ervoor dat je het verwijderen doet op basis van de *id* van het binnenkomende object, niet op basis van de referentie. Deze kan mogelijk namelijk niet hetzelfde zijn, waardoor je niet daadwerkelijk het object verwijderd.
- De *getUniqueld*-methode bepaald welke *id* het hoogst is in de *objects*-List, telt daar vervolgens 1 bij op en returned deze waarde.

5. Voer bovenstaande stappen nogmaals uit voor de DetailDAO, met als aanvulling:

- De *getAllFor*-methode returned alle objecten uit de *objects*-List, maar alleen als deze bij de meegegeven Master-model horen.
- Let op! Mogelijk heb je nu het gevoel dat je voor een deel dubbele code aan het schrijven bent... en dat klopt. Om de practicumopdracht simpel te houden is ervoor gekozen hier niks aan te doen. Voel je echter vrij om te experimenteren en hier alsnog een oplossing voor te verzinnen. De oplossing zal waarschijnlijk wel een richting op gaan die voor de meeste studenten op dit moment te complex zal zijn. Vind je de practicumopdracht van zichzelf al lastig genoeg, maak het jezelf dan nu niet nog lastiger en leef met de dubbele code die je moet schrijven.

6. Implementeer de FakeMasterDAO en FakeDetailDAO. Deze classes gaan je applicatie standaard voorzien van wat hardcoded nep-data, wat handig is tijdens het testen van je applicatie. Zorg ervoor dat je in de *load*-methodes wat objecten aan de eigen DAO toevoegt. Zorg er daarnaast voor dat je Detail-models een *masterId* krijgen toegewezen die verwijst naar de *id* van een Master-model die je hebt toegevoegd, ook deze id's mag je gewoon hardcoden. Vergelijk deze handeling met het instellen van een *Foreign Key* die verwijst naar een *Primary Key*.
  - Let op! De *save*-methode van beide classes kun je leeg laten, gezien we de data nooit daadwerkelijk gaan opslaan.
  - Let op! Je hoeft op dit punt nog niks te doen met de *hoortBijLijst* uit week 1 van de practicumopdracht. Enkele de nieuwe *masterId*, welke je bijvoorbeeld ook *hoortBijLijstId* kan noemen, is voldoende.
7. Breng nu de nodige wijzigingen aan op de controllers en de rest van je applicatie zoals aangegeven in de UML. Zorg er vervolgens voor dat de beide views als volgt gaan werken:
  - Bij het opstarten van een scherm toont de ListView standaard alle bijbehorende models, bijv. de MasterController toont alle Master-models. Gebruik hiervoor databinding in combinatie met de bijbehorende DAO's. Override de *toString*-methode van je models zodanig dat alle data zichtbaar is in de ListViews.
    - Let op! Zorg ervoor dat de DetailController alleen de Details-models in de ListView toont die horen bij de eerder geselecteerde Master-model.
  - Specifiek voor de DetailController: Bij het opstarten van het scherm bevat de verplichte ComboBox uit week 2 van de practicumopdracht een lijst van alle Master-models en wordt standaard de eerder geselecteerde Master-model geselecteerd. Tevens toont de ListView op dit scherm altijd de Detail-models die horen bij het in de ComboBox geselecteerde Master-model.
  - Bij het aanklikken van een model in de ListView worden direct de invoervelden bijgewerkt met de waardes van de geselecteerde model. Dit fungeert als het "bekijken" van de gegevens.
  - Bij het aanklikken van de "Opslaan"-knop wordt, indien een model in de ListView is geselecteerd, de model bijgewerkt met de waardes uit de invoervelden. Dit fungeert als het "bewerken" van bestaande gegevens.
  - Bij het aanklikken van de "Nieuw"-knop wordt de selectie in de ListView gereset en worden alle invoervelden geleegd. Als je hierna op de "Opslaan"-knop drukt worden een nieuw model aangemaakt. Dit fungeert als het "toevoegen" van nieuwe gegevens.

- Bij het aanklikken van de “Verwijderen”-knop wordt het geselecteerde model uit de ListView verwijderd. Vraag de gebruiker om bevestiging met een Ja/Nee-Alert voordat je het model daadwerkelijk verwijderd.
  - Let op! Vergeet bij het verwijderen van een Master-model niet ook de gerelateerde Detail-models te verwijderen, anders blijven er onbruikbare Detail-models in je DAO's hangen.
- Zorg ervoor dat ten alle tijden enkel nuttige knoppen actief zijn, bijv. De “Verwijderen”-knop als er (nog) niks is aangeklikt in de ListView.

Practicumopdracht OOP2 - Lennard Fonteijn

ComboBox: Id: 1 (TodoLijst)

TextField: Test 2

TextArea:

DatePicker: 28-02-2020

CheckBox: ☒ (boolean)

Opslaan

(TodoRegel)  
Id: 1  
Test 1

(TodoRegel)  
Id: 2  
Test 2

(TodoRegel)  
Id: 5  
Test 3

Nieuw Verwijderen Terug naar overzicht

*Afbeelding: Voorbeeld van een TodoRegelView na het afronden van alle stappen en het aanklikken van een model in de ListView.*

## Checklist

- ☐ De DAO's zijn geïmplementeerd zoals aangegeven in de UML.
  - ☐ Er is een DAO-interface aanwezig.
  - ☐ Er is een abstracte MasterDAO- en DetailDAO-class aanwezig met juiste naamgeving.
  - ☐ Er is een FakeMasterDAO- en FakeDetailDAO-class aanwezig met enkel een *save*- en *load*-methode.
  - ☐ Alle interfaces en classes zitten in een *data*-package.
- ☐ De DAO's vertonen het gedrag zoals beschreven in het stappenplan:
  - ☐ De *getAll*-methode geeft een kopie van de *objects*-List terug.
  - ☐ De *getAllFor*-methode geeft een List terug met enkel de relevante Detail-models voor de aangegeven Master-model.
  - ☐ De *get*-methode geeft de gevraagde model terug op basis van de gegeven *id*.
  - ☐ De *addOrUpdate*-methode voegt een object toe indien deze nog niet bestaat, of vervangt deze indien de *id* al aanwezig is.
  - ☐ De *remove*-methode verwijdert een object op basis van de gegeven *id*.
  - ☐ De *getUniqueld*-methode bepaald de hoogste *id* in de *objects*-List, telt er 1 bij op en geeft deze terug.
- ☐ De Fake-DAO's voegen middels hun *load*-methode nep-data toe.
- ☐ De models hebben een *toString*-methode die alle data bevat.
- ☐ De views vertonen het gedrag zoals beschreven in het stappenplan:
  - ☐ De ListView's tonen enkel de relevante models en zijn altijd up-to-date met de laatste wijzigingen.
  - ☐ De verplichte ComboBox van week 1 toont alle Master-models.
  - ☐ De selectie van de ComboBox beïnvloed welke Detail-models er zichtbaar zijn in de ListView.
  - ☐ Het selecteren van een model in de ListView werkt gelijk alle invoervelden bij.
  - ☐ Het is mogelijk om zowel bestaande models te bewerken, als nieuwe models toe te voegen.
  - ☐ Verwijderen geeft de gebruiker de kans om dat te annuleren middels een *Ja/Nee-Alert*.
  - ☐ Bij het verwijderen van een Master-model worden ook relevante Detail-models verwijderd.
  - ☐ Er zijn ten alle tijden enkel relevante knoppen actief.
- ☐ Het project bevat geen compile-errors.
- ☐ Het project is gecommit en gepushed naar GitLab.