

# HO GENT

Grafische userinterface - Oefeningen

# Table of Contents

1. Oefening Rekening – DEEL 1 .....	1
2. Oefening FXRodeDraad – DEEL 2 .....	3
3. Oefening Voorwerp .....	5
4. Oefening Simon .....	8

# 1. Oefening Rekening – DEEL 1

- Gegeven:
  - zie Chamilo: H05\_Oef1\_Rekening\_Start
  - breid Oefening Rekening uit het hoofdstuk "Polymorfisme en interfaces" uit met een grafische user interface.
- Gevraagd: vul de gui-klasse BeheerskostenScherm aan volgens de UML en de richtlijnen.
  - Een combobox, gevuld met titels van items, wordt weergegeven. Een item is een object van een implementatie-klasse van de interface Beheerskosten.
  - De gebruiker kan een item selecteren uit de combobox om de details van het item en de beheerskosten weer te laten geven in een textarea en een textfield
  - Console user interface (zie H02):

zichtrekening, houder = Michiel  
Details: ZichtRekening met rekeningnummer 000-0000004-04  
staat op naam van Michiel  
en bevat 400,00 euro. Max krediet onder nul = -2000,00  
Kost: 20,00

kluisnr = 59, houder = Lander  
Details: kluisnr = 59, houder = Lander  
Kost: 29,00

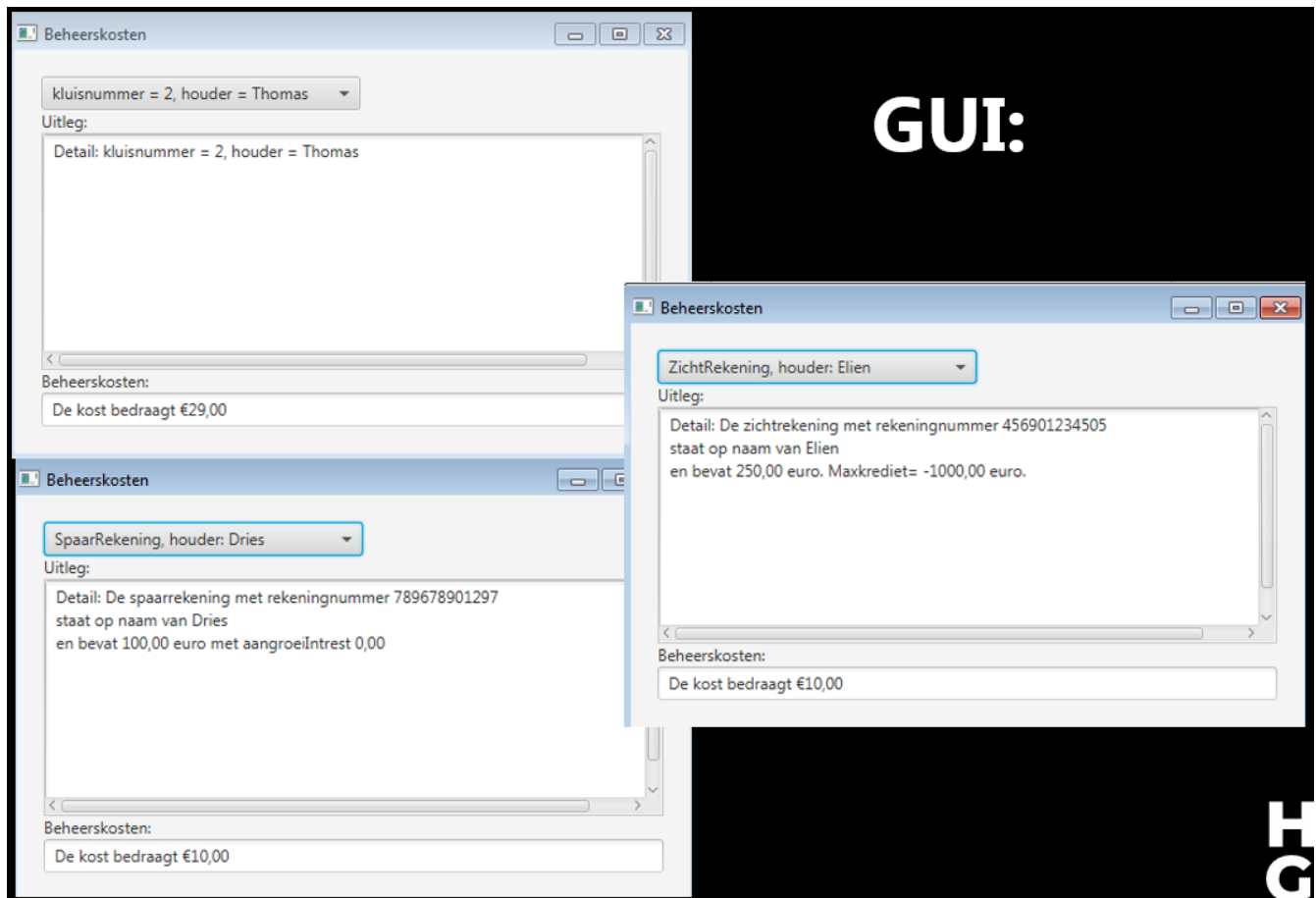
zichtrekening, houder = Stef  
Details: ZichtRekening met rekeningnummer 000-0000003-03  
staat op naam van Stef  
en bevat 300,00 euro. Max krediet onder nul = -1000,00  
Kost: 10,00

spaarrekening, houder = Sebastien  
Details: SpaarRekening met rekeningnummer 000-0000002-02  
staat op naam van Sebastien  
en bevat 200,00 euro. Aangroeiintrest = 2,00%  
Kost: 5,00

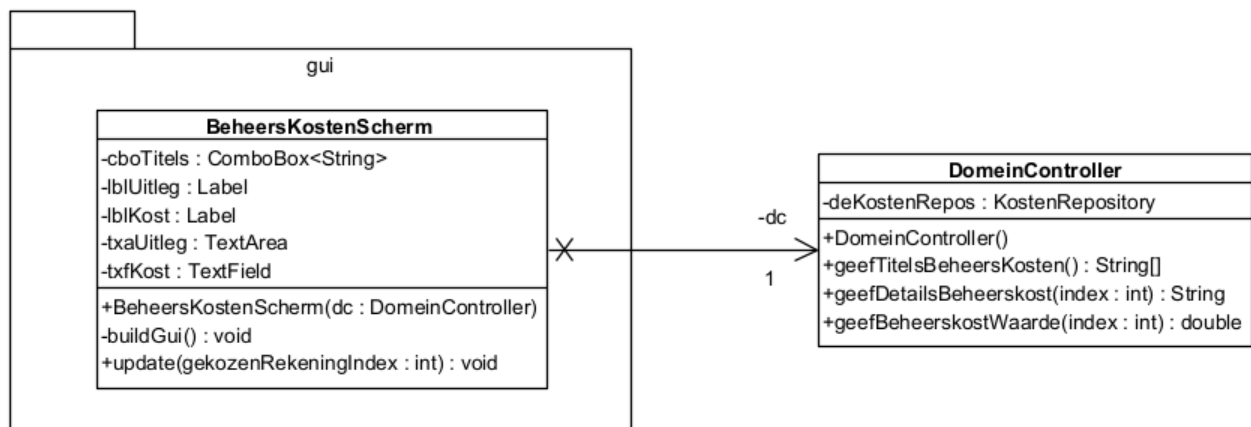
spaarrekening, houder = Toon  
Details: SpaarRekening met rekeningnummer 000-0000001-01  
staat op naam van Toon  
en bevat 100,00 euro. Aangroeiintrest = 2,00%  
Kost: 10,00

kluisnr = 58, houder = Dylan  
Details: kluisnr = 58, houder = Dylan  
Kost: 29,00

- Grafische user interface (gevraagd):



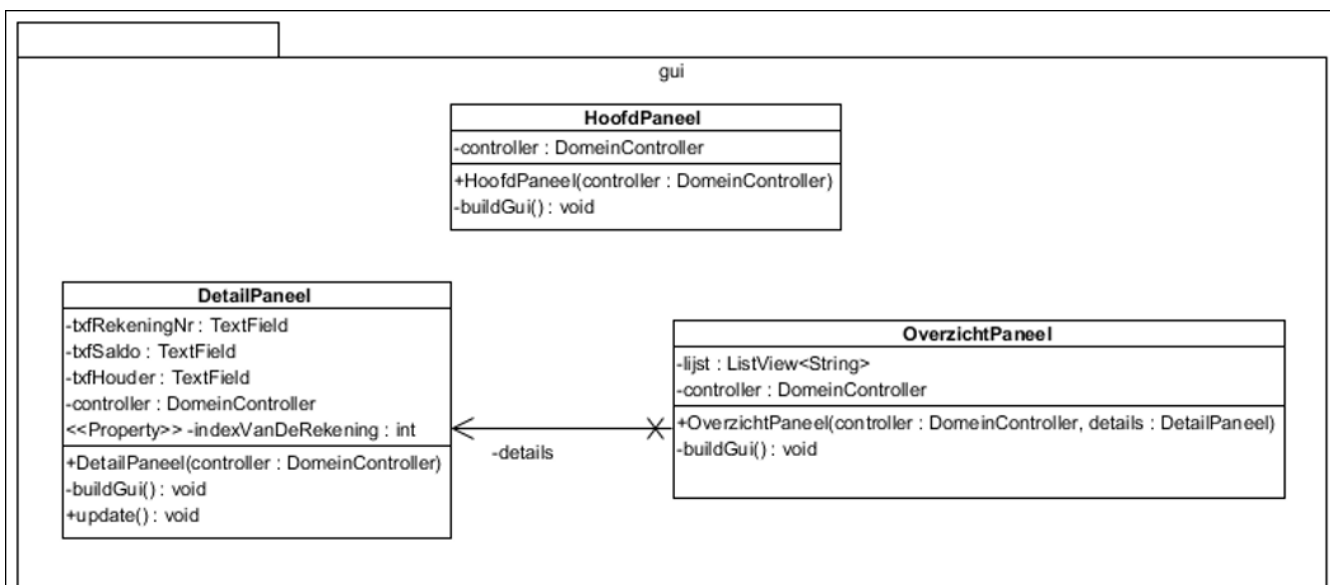
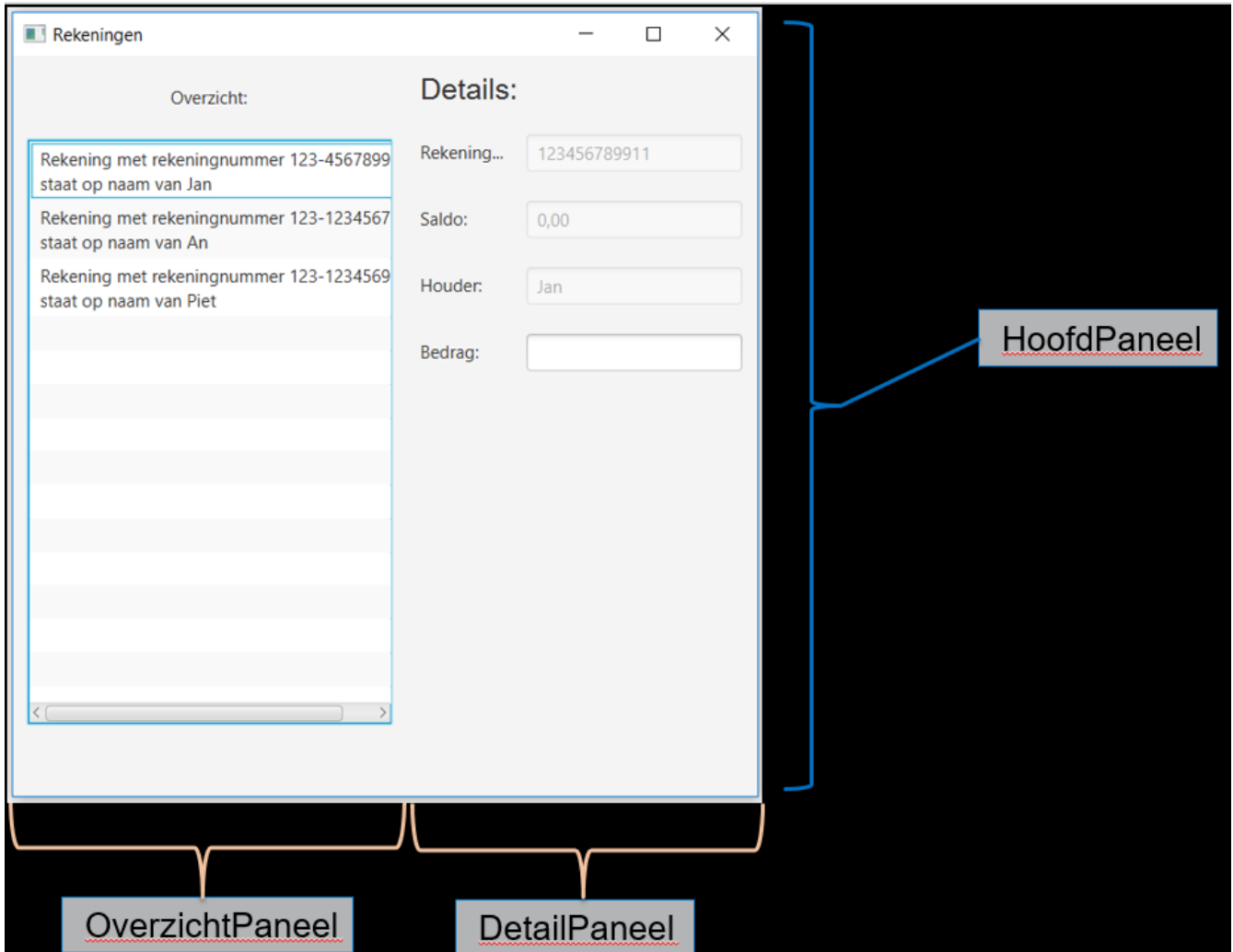
- UML (gegeven):



## 2. Oefening FXRodeDraad – DEEL 2

- Gegeven:
  - Het project H05\_Oef2\_Rekening\_Start met daarin een domeinklasse Rekening (zie Chamilo) bestaande uit:
    - Attributen: rekeningNr (long), saldo (double) en houder (String)
    - Methoden: 2 constructoren, get- en setmethoden, stortOp, haalAf, schrijfBedragOverNaar en toString
- Gevraagd:

- Maak een overzicht van alle bestaande rekeningnummers in een OverzichtPaneel
- Maak een DetailPaneel dat de details toont van de geselecteerde rekening. Wanneer een andere rekening geselecteerd wordt in het OverzichtPaneel dan wordt het DetailPaneel aangepast.
- Via het veld “bedrag” op het DetailPaneel moet je geld kunnen storten op de geselecteerde rekening.

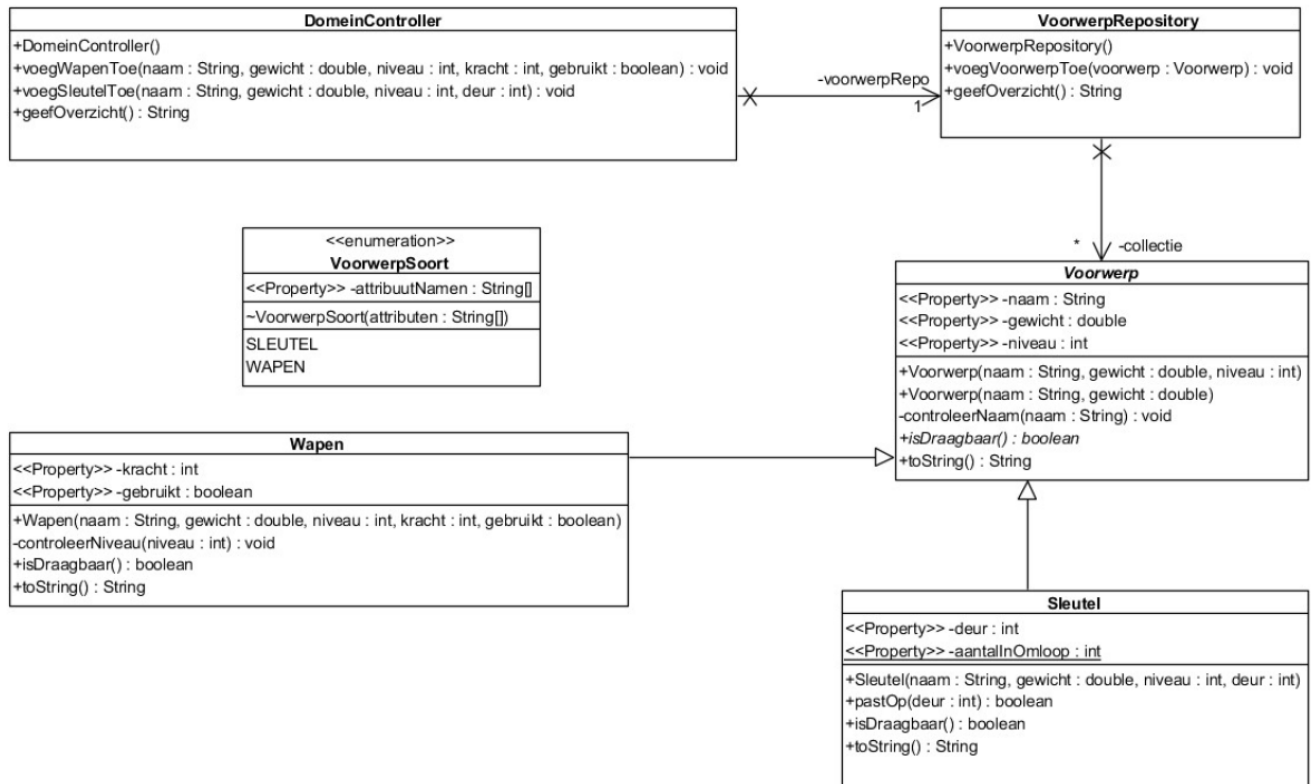


### 3. Oefening Voorwerp

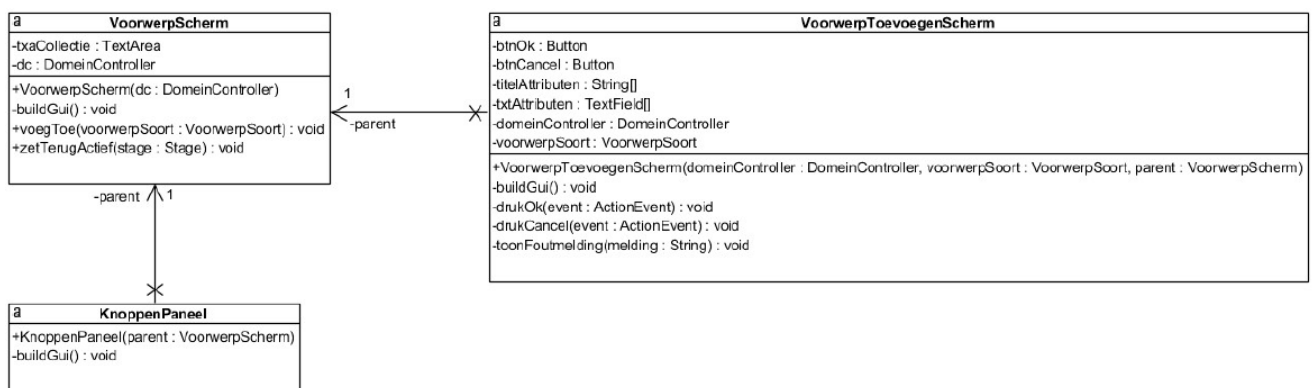
We gebruiken een afgeslankte versie van de vroegere opgave ivm Voorwerpen (zie Chamilo voor startproject H05\_Oef3\_Voorwerp\_Start). We voegen er enkel nog een enumklasse aan toe (zie verder opgave).

We bouwen een grafische user interface.

#### Package domein



#### Package gui



- Het domein breiden we uit met een enum-klasse (zie Chamilo):

```

package domein;

public enum VoorwerpSoort
{
    SLEUTEL(new String[] { "naam", "gewicht", "niveau", "deur" }),
    WAPEN(new String[] { "naam", "gewicht", "niveau", "kracht", "gebruikt" });

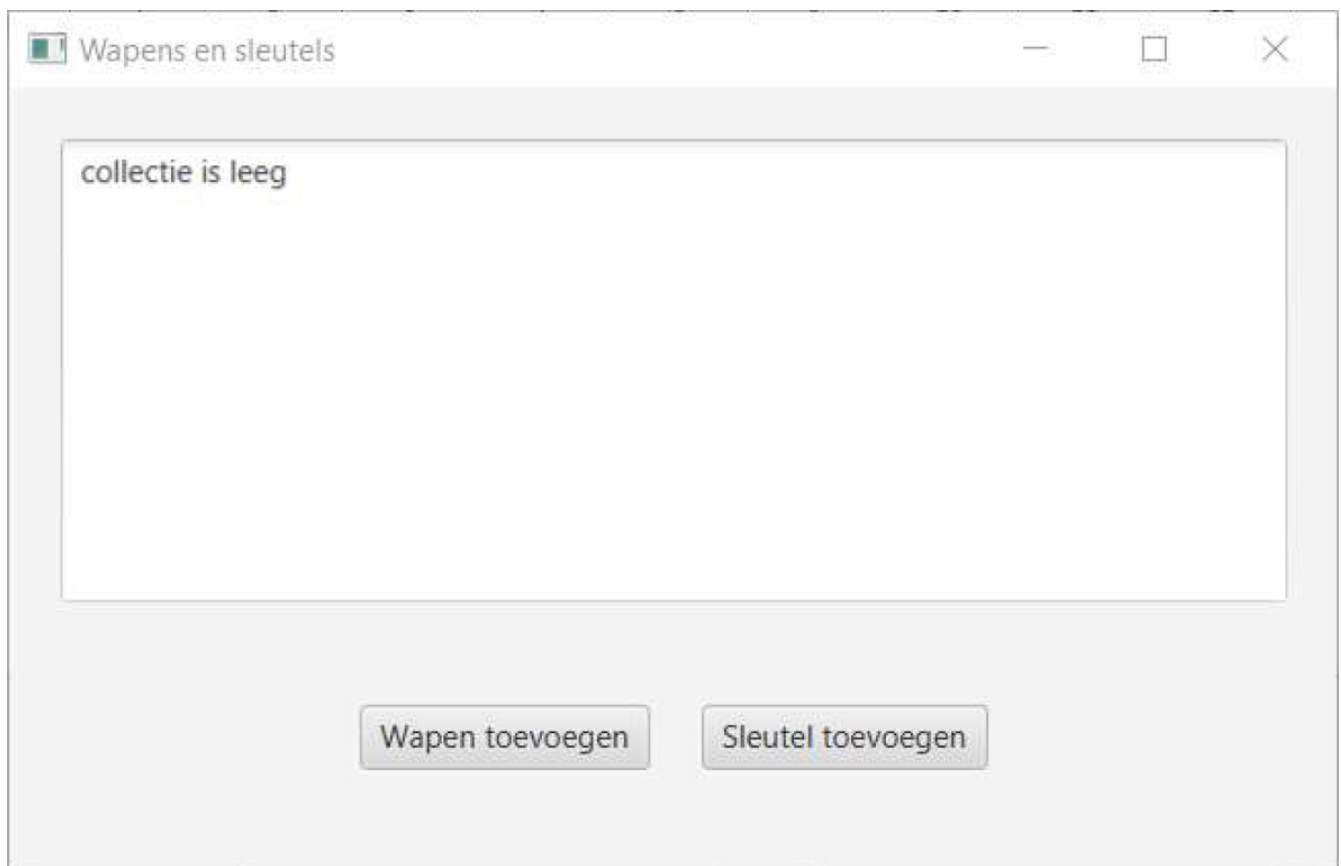
    private String[] attribuutNamen;

    VoorwerpSoort(String[] attributen) {
        attribuutNamen = attributen;
    }

    public String[] getAttribuutNamen() {
        return attribuutNamen;
    }
}

```

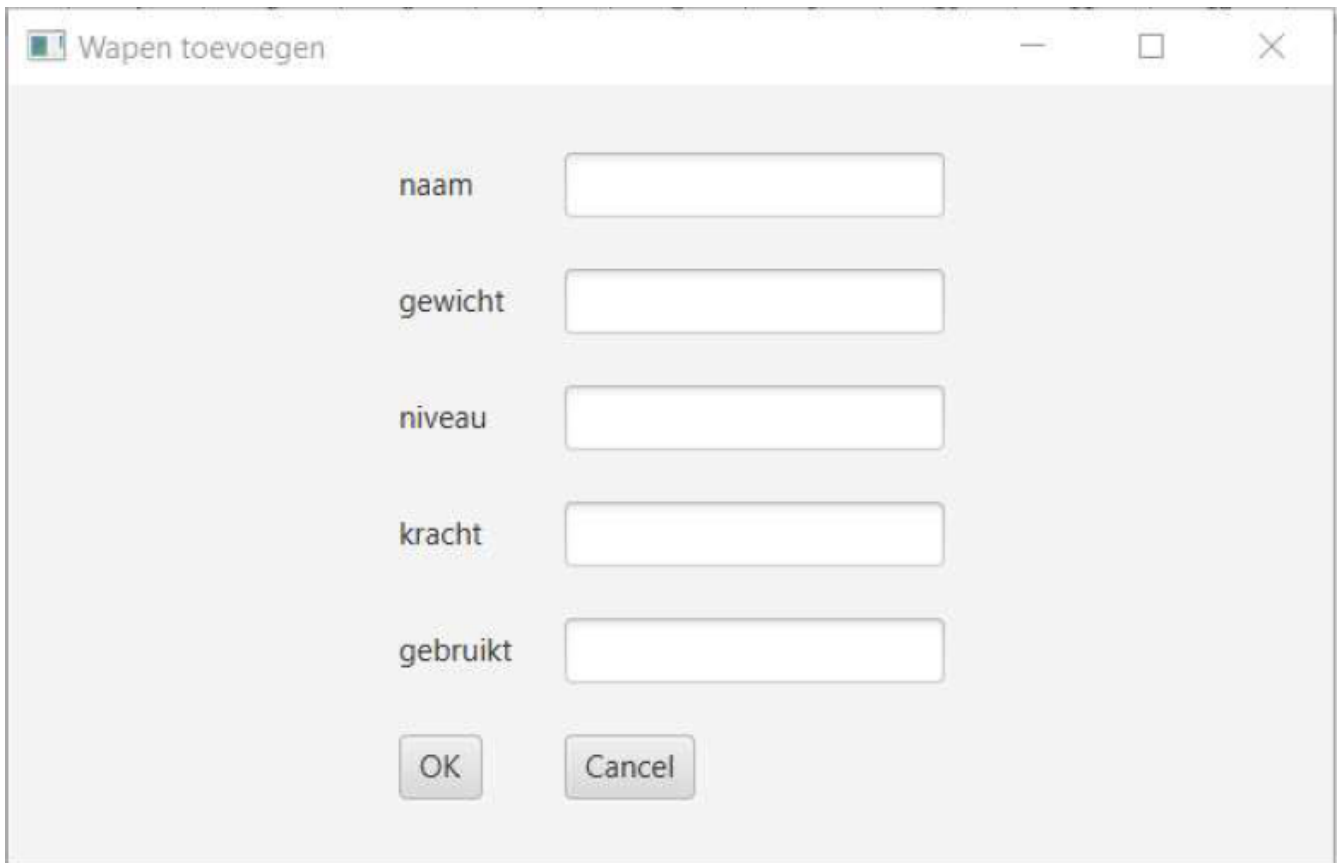
- Ontwerp nu schermen om wapens en sleutels toe te voegen aan je collectie. De huidige collectie wordt weergegeven.
- Het VoorwerpScherm (VBox) bestaat uit een TextArea en een KnoppenPaneel eronder.
- VoorwerpScherm:



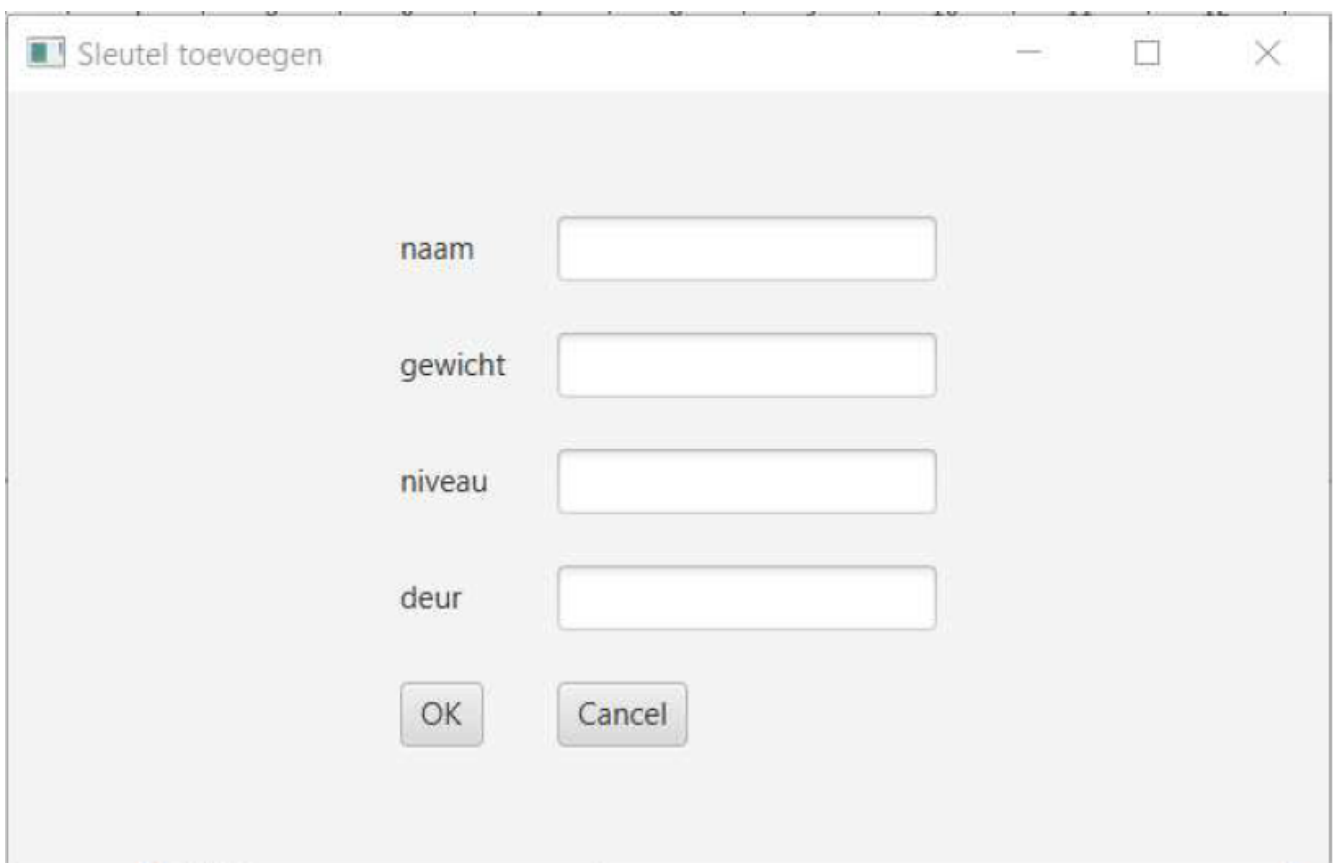
- Het KnoppenPaneel heeft een HBox-layout.
- Het volgende scherm (VoorwerpToevoegenScherm) verschijnt als op de knop "Sleutel



toevoegen" of "Wapen toevoegen" wordt geklikt. Dit scherm verschijnt op dezelfde Stage.



The screenshot shows a Windows-style dialog box titled "Wapen toevoegen". It contains five text input fields, each preceded by a label: "naam", "gewicht", "niveau", "kracht", and "gebruikt". At the bottom of the dialog are two buttons: "OK" and "Cancel".



The screenshot shows a Windows-style dialog box titled "Sleutel toevoegen". It contains four text input fields, each preceded by a label: "naam", "gewicht", "niveau", and "deur". At the bottom of the dialog are two buttons: "OK" and "Cancel".

- Met behulp van de enum-klasse zorg je ervoor dat het scherm om een sleutel toe te voegen en het scherm om een wapen toe te voegen, met één klasse kunnen gedefinieerd worden!
- TIP: Het aantal **attribuutNamen** bepaalt het aantal rijen (+ 1 voor de buttons) in de GridPane!

- Mogelijk resultaat na het toevoegen van 3 Voorwerpen (1 Wapen en 2 Sleutels) op het VoorwerpScherm:



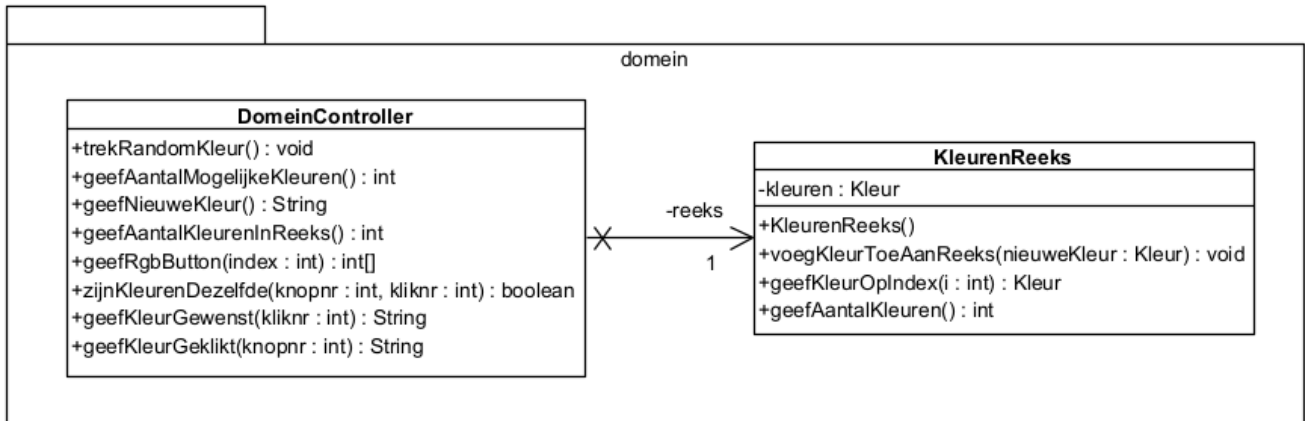
- Werk met een Alert-venster in de catchers (via de methode toonFoutmelding):



## 4. Oefening Simon

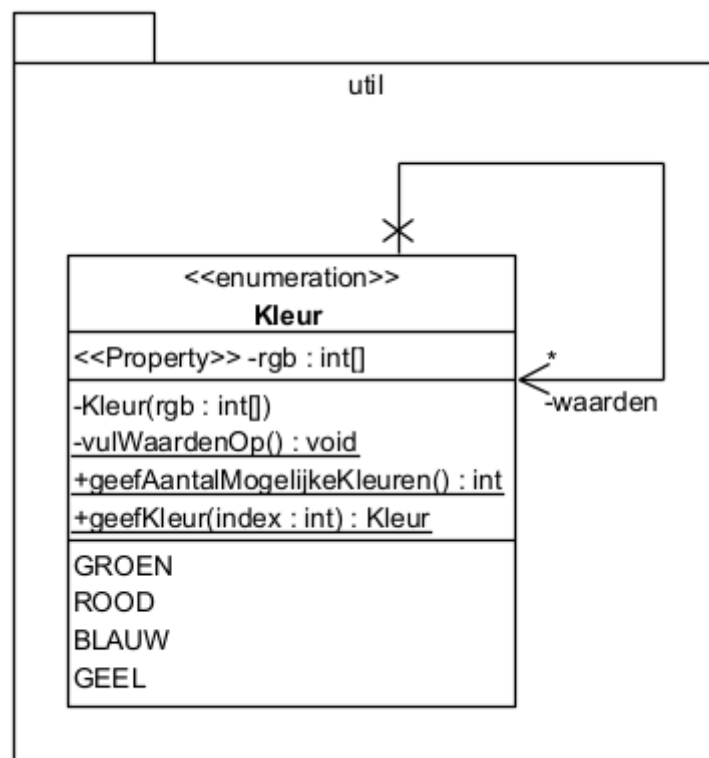
Maak een applicatie voor het spel "Simon says". Hierbij moet de speler een kleurenreeks onthouden die telkens uit één extra kleur bestaat. Het spel eindigt indien de speler de reeks niet meer kan herhalen.

De domeinlogica van het spel is gegeven en voldoet aan volgende UML:



Hierbij wordt ook gebruik gemaakt van een enumeratie `Kleur` die de mogelijke kleuren kan bijhouden. In het voorbeeld zijn dat de 4 kleuren GROEN, ROOD, BLAUW en GEEL. De enumeratie houdt ook een `Kleurarray` (genaamd `waarden`) bij die in het begin naar null verwijst. De methode `vulWaardenOp` checkt of de array nog steeds naar null verwijst en vult in dat geval de array op met alle mogelijke kleuren.

Verdere informatie over de util-package vind je in onderstaande UML:

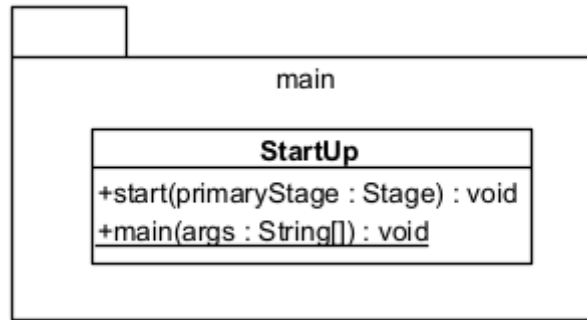


Ook hiervan is de code gegeven in het startproject (H05\_Oef4\_Simon\_Start) dat je op Chamilo kan vinden.

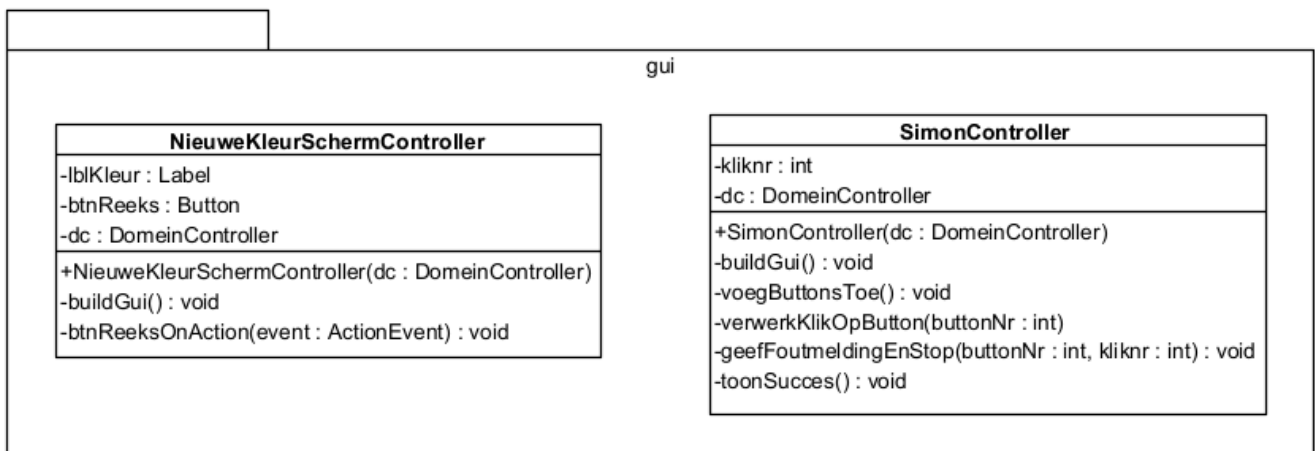
## Gevraagd

Vul de packages `main` en `gui` verder aan om tot een werkend "Simon says"-spel te komen.

In de `main`-package komt een klasse `Startup` die voldoet aan volgende UML:



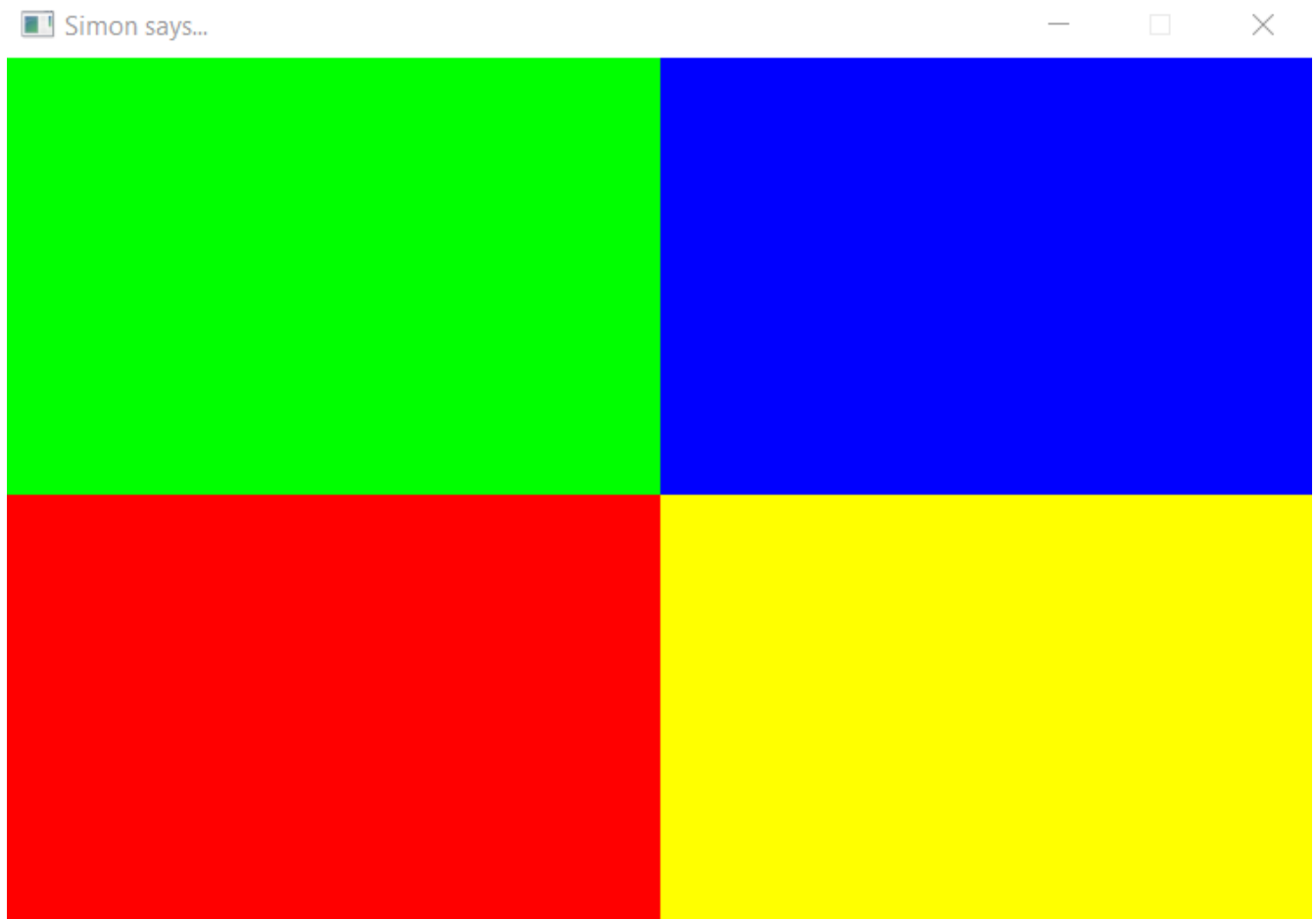
Bij het opstarten van het programma moet een NieuweKleurScherm verschijnen dat er uit ziet zoals hieronder beschreven en afgebeeld. Het scherm mag niet groter of kleiner gemaakt kunnen worden. In de gui-package komen twee klassen voor (naam kan afwijken naargelang je al dan niet SceneBuilder gebruikt), waarvan je hieronder de UML vindt:



Op het NieuweKleurScherm staan 2 labels: eentje met de vaste tekst "De nieuwe kleur is..." en

eentje (lblKleur) met een random gekozen Kleur uit de enumeratie erop. Bij het aanmaken van het scherm wordt deze willekeurige Kleur bepaald en wordt deze ook toegevoegd aan de KleurenReeks.

Ook is er een Button (btnReeks) met daarop de tekst "Geef de reeks". Als je daarop klikt, komt de methode btnReeksOnAction in actie, die ervoor zorgt dat het tweede scherm te voorschijn komt.



Het tweede scherm is het spel Simon zelf en dit bestaat dus uit 4 kleurvakken (zorg voor een mogelijkheid om dit uit te breiden tot om het even welk even aantal vakken) waarvan er telkens 2 op een rij staan. Deze vakken zijn eigenlijk knoppen waar de gebruiker kan op klikken. Om deze knoppen te genereren gebruik je de methode `voegKnoppenToe`.

Maak in deze methode een 1-dimensionale array van Buttons die elk hun eigen kleur krijgen via de rgbwaarde die je uit de enumeratie opvraagt en waarmee je via onderstaande een Background maakt, die je dan met de methode `setBackground` kunt instellen. Vergeet ook niet de button op de juiste plaats in de GridPane te zetten.

```
Color kleur = Color.rgb(rgbButton[0], rgbButton[1], rgbButton[2]);  
Background achtergrond = new Background(new BackgroundFill(kleur, null, null));
```

Wanneer de gebruiker op één van de knoppen klikt komt de methode `verwerkKlikOpButton` in actie. Let op: je kan hiervoor geen method reference gebruiken, aangezien de eventHandler een ActionEvent als parameter heeft, terwijl de methode `verwerkKlikOpButton` een nummer meekrijgt bij die het nummer van de button, te tellen vanaf 0 en per rij van links naar rechts oplopend, voorstelt. De rechterbutton van de tweede rij heeft dus bijvoorbeeld nummer 3. Om dit nummer vanuit de

methode `voegKnoppenToe` te kunnen doorgeven, moet het als final gedeclareerd worden.

Aan de hand van het nummer dat als argument van de methode `verwerkKlikOpButton` meegegeven, kunnen we weten welke Kleur de gebruiker geklikt heeft. In de klasse `SimonScherm` wordt ook bijgehouden in de variabele `kliknr` hoeveel keer de gebruiker al geklikt heeft. Zo kunnen we weten welke Kleur uit de reeks aangeklikt had moeten worden. Door dus het `buttonnr` en het `kliknr` door te geven aan de `DomeinController` kan die bepalen of de juiste kleur werd aangeklikt. Indien dit niet het geval is, wordt de methode `geefFoutmeldingEnStop` aangeroepen, anders wordt het aantal klikken verhoogd en wordt gecheckt of we reeds aan de laatste klik zijn toegekomen. Indien ja, dan wordt de methode `toonSucces` aangeroepen.

In `geefFoutmeldingEnStop` wordt een popupvenster getoond die aangeeft welke Kleur de gebruiker heeft aangeklikt en welke hij had moeten aanklikken. Hiervoor roepen we opnieuw de betreffende methodes van de `DomeinController` aan en gebruiken we de benodigde argumenten. Tevens wordt getoond hoeveel kleuren er al in de reeks zaten. Het woord "kleur" wordt hierbij in het enkelvoud of meervoud getoond. Daarna stopt het programma.

Bijvoorbeeld:

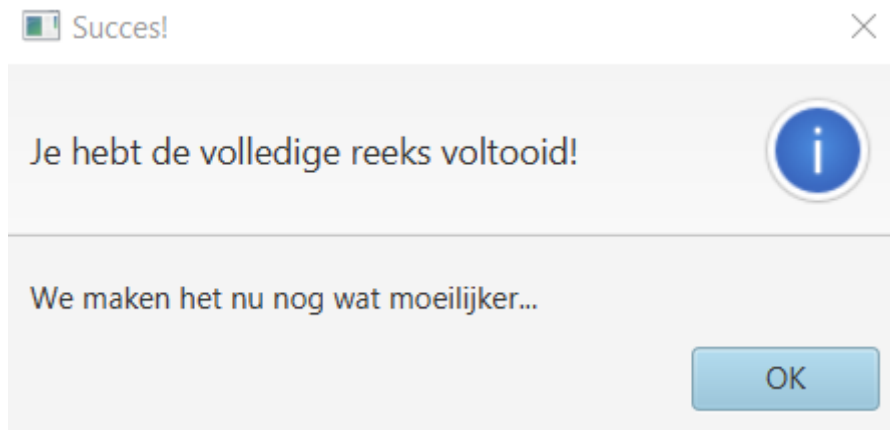


of

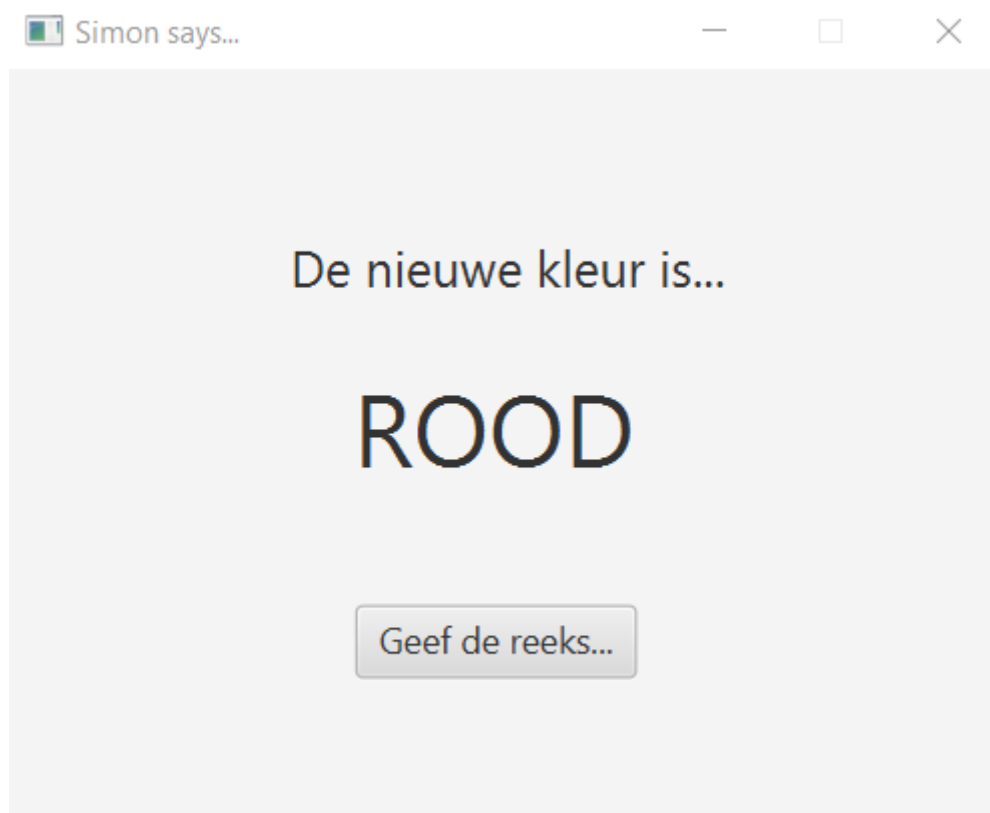


In `toonSucces` wordt ook een popupvenster getoond, maar dan met de mededeling dat je nog steeds goed bezig bent. Als je deze popup wegklikt, dan krijg je opnieuw het eerste scherm (`NieuweKleurScherm`) te zien met de volgende random kleur die aan de reeks wordt toegevoegd en die je moet onthouden.

Bijvoorbeeld:



of



De volgende keer dat je dan op het scherm met de kleuren komt, moet je (in de juiste volgorde!) op twee buttons klikken om door te mogen gaan. En zo verder...