

# JAVA ADVANCED 2020-2021

NELE CUSTERS

HOGESCHOOL PXL







# Inhoudsopgave

1	Aan de slag	6
2	Ontbrekende klassen en enums toevoegen	6
2.1	Enum CreditCardType . . . . .	9
2.2	Klasse PaymentInfo . . . . .	9
2.3	Enum StreamingPlan . . . . .	9
2.4	Klasse Profile . . . . .	10
2.5	Klasse Account . . . . .	10
3	Optioneel: Sterkte van het paswoord	10
4	De toepassing uitvoeren	11
1	Content hiërarchie	12
2	Klasse Account en associaties	14
2.1	CreditCardType . . . . .	14
2.2	CreditCardNumber . . . . .	14
2.3	PaymentInfo . . . . .	14
2.4	Profile . . . . .	16
2.5	Account . . . . .	16
3	AccountRepository	17
1	Deeltaak 1: Domino	1
1.1	De spelregels . . . . .	1
1.2	De implementatie . . . . .	3
1.3	Voorbeeld output van een spel . . . . .	3
1.4	Gedeeltelijk klassendiagram . . . . .	6
2	Deeltaak 2: Afbeelding met domino-steentjes	7
3	Deeltaak 3: Kunst met domino-steentjes	8







## 1 Aan de slag

---

De github repository <https://github.com/custersnele/JavaAdvStreamingServiceV1> bevat een grafische userinterface (GUI) voor de streaming service. De GUI is geschreven in JavaFX.

Fork dit project zodat je een eigen repository hebt waarmee je aan de slag kan gaan. Je kan dit project daarna openen in IntelliJ IDEA. Dit project is een Maven project. Omdat we een JavaFX en nog enkele andere libraries nodig hebben, is het interessant om Maven te gebruiken. Maven vergemakkelijkt het beheren van 3rd party libraries. Maven biedt nog veel meer dan enkel 3rd party libraries beheren in een project, maar dat valt buiten de scope van deze cursus. De 3rd party libraries die we gaan gebruiken vind je terug tussen de `<dependencies>` tags in het bestand `pom.xml`. Dit bestand, het project object model (POM), bevat alle nodige configuratie voor Maven.

Merk ook op dat Maven zijn eigen default mappenstructuur heeft.

Je ziet dat er nog veel compiler errors zijn in ons project. De code om de GUI te tonen en aan te sturen is reeds beschikbaar, maar we missen de klassen die we afgelopen weken reeds gedeeltelijk hebben ontwikkeld.

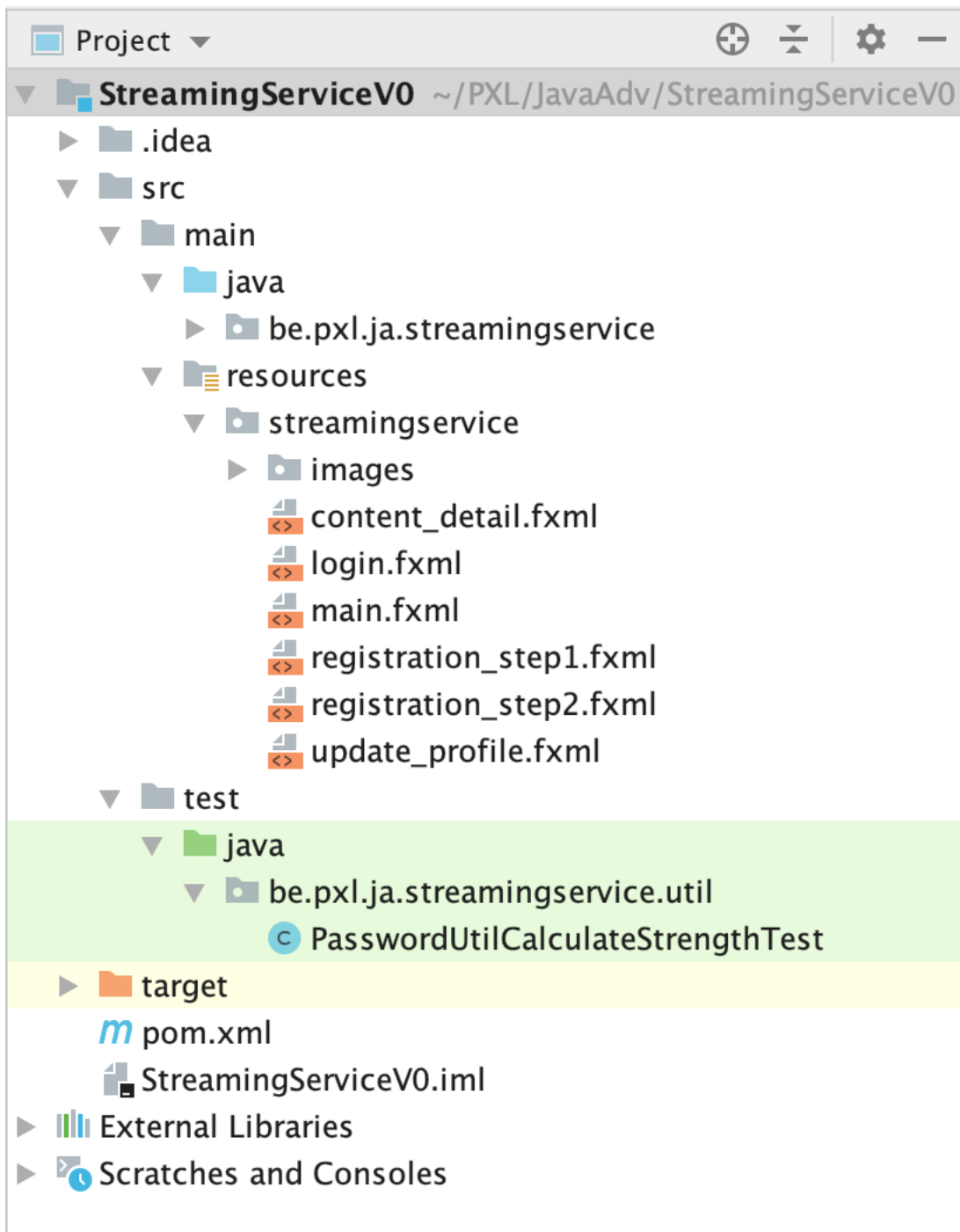
**Theorem 0.1** De klassen die je tot nu toe hebt ontwikkeld voor de streaming service voeg je toe in het package `be.pxl.java.streaming.service.model`. Kopieer ook de bijhorende unit testen naar de juiste testfolder.

We zullen nu alle klassen 1 voor 1 overlopen en je nog moet aanpassen of toevoegen. Een volledig klassendiagram vind je aan het einde van het hoofdstuk.

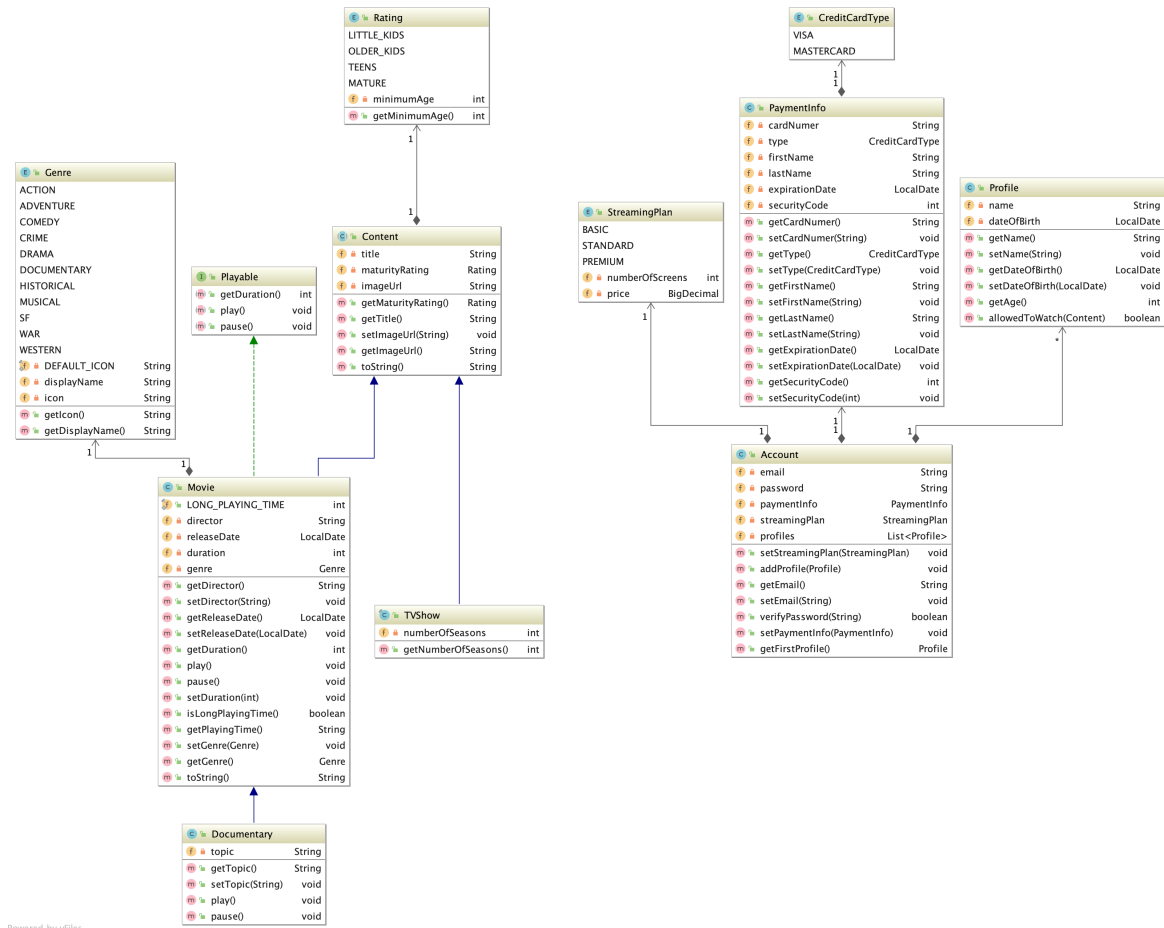
## 2 Profile

---

We voegen in de klasse `Profile` een extra eigenschap toe: de eigenschap `avatar` (`String`). Pas ook de constructor van de klasse `Profile` aan. De constructor heeft nu 2 parameters: `name`



Figuur 1: Mappenstructuur van het maven project



Figuur 2: Klassen van de streaming service



en avatar. Voor een nieuw Profile-object is de geboortedatum null (niet ingevuld).

In de klasse Profile voorzie je drie verzamelingen voor Content-objecten: recent bekeken content (recentlyWatched), het materiaal wat momenteel wordt bekeken (currentlyWatching) en een verzameling met te bekijken content (My List of wachtlijst). Kies een gepaste klasse uit het Java Collection framework voor elke van deze verzamelingen. De methode startWatching in de klasse Profile zorgt ervoor dat de content **vooraan** in de verzameling currentlyWatching wordt toegevoegd. (Zorg er ook voor dat elk Content-object maar maximum 1 keer in de verzameling kan voorkomen.) De methode finishWatching() zorgt ervoor dat het Content-object niet meer voorkomt in de verzameling currentlyWatching, maar wel vooraan de verzameling recentlyWatched wordt toegevoegd. (zie ook oefening 4.3).

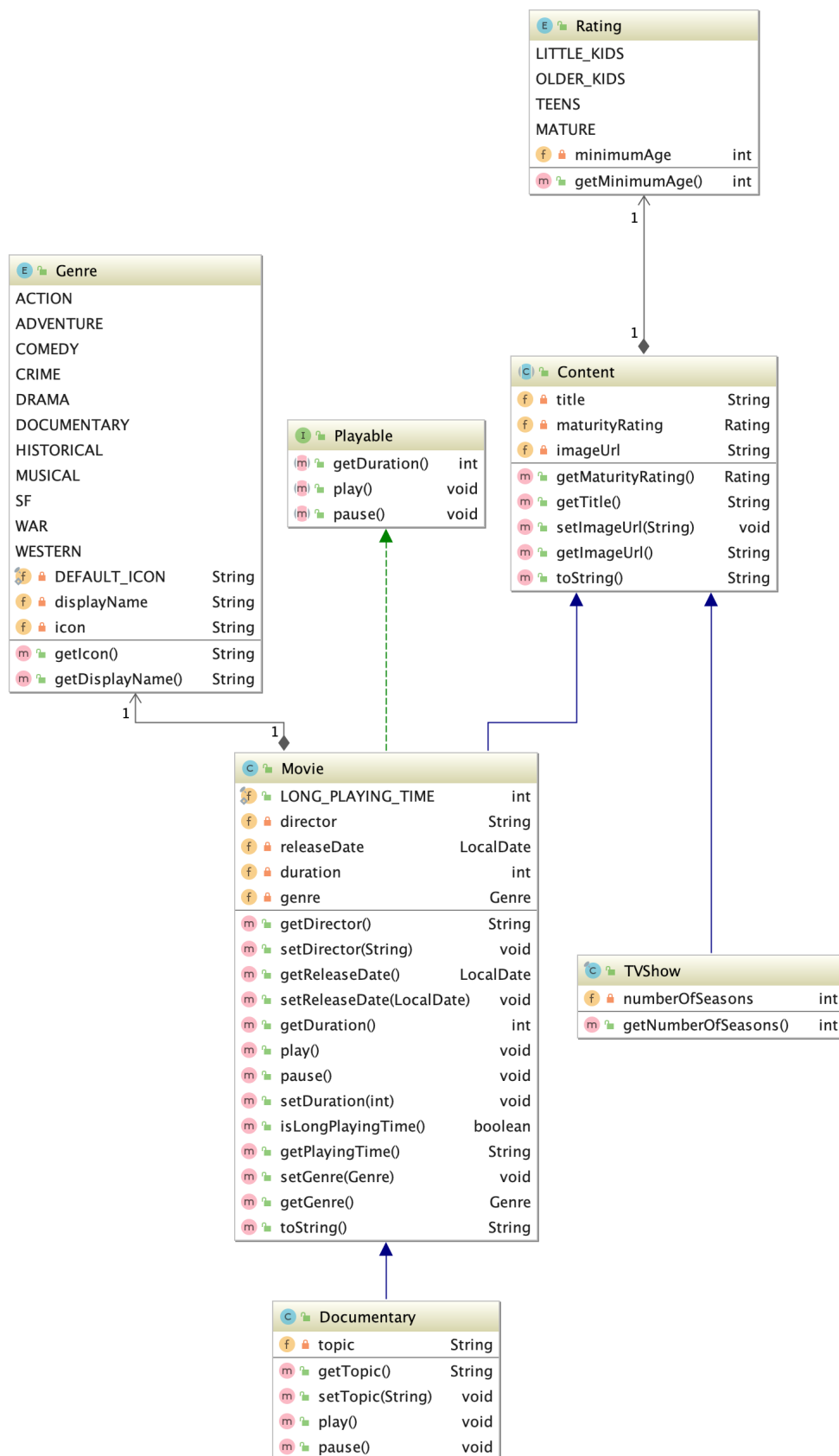
In de klasse Account kan een gebruiker, afhankelijk van het gekozen StreamingPlan, kijkprofielen toevoegen aan een Account. Gebruik een HashMap om deze Profile-objecten bij te houden. Hou rekening met het gekozen StreamingPlan als je een Profile-object toevoegt. Voorzie een unchecked exception TooManyProfilesException() die je opgooit als er geen extra Profile-objecten meer toegevoegd mogen worden. Gebruik de naam (name) van een Profile-object als key. Voorzie in de klasse Account een methode *Profile* `getProfile(String name)`. Schrijf unit testen.

Er kan ook steeds materiaal bij de wachtlijst toegevoegd of verwijderd worden. De methoden `addToMyList()` en `removeFromMyList()` moeten hiervoor gebruikt worden. Zorg dat er nooit Content-objecten meerdere keren in deze wachtlijst kunnen voorkomen.

### 3 Content hiërarchie

---

De content hiërarchie is reeds volledig geïmplementeerd in de vorige versie. Hier volgt nog eens het klassendiagram voor de volledigheid.



Figuur 3: Klassen van de Content-hiërarchie

## 4 Klasse Account en associaties

---

### 4.1 CreditCardType

We hebben 2 type kredietkaarten die we aanvaarden: VISA en MASTERCARD. Nummers van visa-kaarten starten steeds met een “4”, nummers van Mastercard-kaarten starten altijd met een “5”. Deze informatie hou je ook bij in de enum-waarden.

### 4.2 CreditCardNumber

De klasse CreditCardNumber hebben we reeds grotendeels geïmplementeerd in het hoofdstuk Exceptions. Objecten van de klasse CreditCardNumber mogen nooit ongeldige gegevens bevatten. Wanneer je een object van de klasse CreditCardNumber probeert aan te maken met ongeldige gegevens zal er een IllegalArgumentException gegooid worden.

Er wordt steeds gecontroleerd dat een kaartnummer bestaat uit 16 cijfers en het controlegetal (cvc) uit 3 cijfers. Aan de hand van het kaartnummer kan ook het type van de kredietkaart afgeleid worden.

### 4.3 PaymentInfo

Pas de klasse PaymentInfo aan zoals weergegeven in het klassendiagram. In deze klasse gebruik je de klasse CreditCardNumber. Bij de setter setExpirationDate controleer je dat de kredietkaart nog minstens één maand geldig zal zijn. Indien dit niet het geval is, zal de InvalidDateException opgegooid worden. De InvalidDateException is een runtime exception. Meer info vind je ook bij oefening 3.7 (hoofdstuk Exception handling).



Figuur 4: Klassen Account en associaties



## 4.4 Profile

Een profile heeft 3 eigenschappen: een naam (`name`), geboortedatum (`dateOfBirth`) en avatar. De avatar wordt gebruikt om een profiel-afbeelding te tonen. Bij de geboortedatum controleren we dat deze niet in de toekomst ligt (zie ook oefening 3.9). Indien een geboortedatum in de toekomst wordt gegeven, gooi je een `InvalidDateException`.

De klasse `Profile` voorziet 2 constructoren. Een eerste constructor met 2 eigenschappen: `name` en `avatar`. Een tweede constructor voorziet 1 eigenschap: `name`, en geeft avatar de waarde `"profile1"`.

I

## 4.5 Account

Ook in de klasse `Account` werden de afgelopen weken enkele aanpassingen uitgevoerd. Zo zal je in de constructor extra validaties moeten uitvoeren. De eigenschappen `email` en `password` zijn verplicht. Gooi een `IllegalArgumentException` indien de parameters voor deze eigenschappen `null` of een lege string zijn. In de constructor wordt er ook een eerste kijkersprofiel aangemaakt met de naam `"Profile 1"`. `StreamingPlan.BASIC` wordt als default waarde voor een nieuw aangemaakt account toegekend.

De methode `setPassword` gaat altijd controleren dat een gekozen wachtwoord een minimum sterkte van 5 heeft. De `PasswordUtil`, waarmee je de sterkte van een wachtwoord kan berekenen, is reeds aanwezig. Gooi opnieuw een `IllegalArgumentException` indien het paswoord niet de gewenste sterkte heeft.

Bij een account kunnen een aantal kijkersprofielen aangemaakt worden. Het aantal kijkersprofielen is afhankelijk van het gekozen `StreamingPlan`. Controleer in de methode `addProfile()` dat het aantal profielen het aantal toegelaten profielen van het gekozen `StreamingPlan` niet overschrijdt. Je gooit een `TooManyProfilesException` wanneer het toegelaten aantal profielen wordt overschreden (zie ook oefening 4.6).

Zorg er ook voor dat de methode `getProfiles()` de `Profile`-objecten altijd gesorteerd op naam teruggeeft.

Extra: zoek eens op hoe je kan controleren of een opgegeven e-mailadres wel degelijk een correct formaat heeft. Indien het e-mailadres een ongeldig formaat heeft gooi je een `IllegalArgumentException`.

## 5 AccountRepository

---

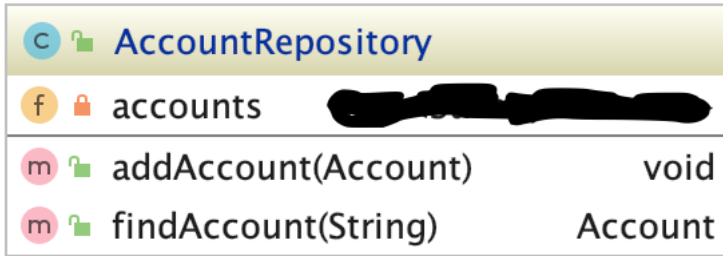
In het package `be.pxl.ja.streaming.service.repository` maak je een klasse `AccountRepository`. Deze klasse bevat een verzameling van alle accounts die worden aangemaakt voor onze streaming service. Je mag kiezen welke klasse uit het Java Collection framework je gebruikt, maar zorg dat je eenvoudig een `Account`-object kan terugvinden aan de hand van een gegeven e-mailadres.

De klasse `AccountRepository` bevat volgende functies:

- `void addAccount(Account newAccount)`: voegt een nieuw account toe aan de `AccountRepository`. Indien er al een account bestaat met hetzelfde e-mailadres als dat van het nieuwe account wordt het nieuwe account niet toegevoegd, maar gooi je een `DuplicateEmailException` met als boodschap dat er reeds een account bestaat met het opgegeven e-mailadres.

- *Account findAccount(String email)*: geef het Account-object dat behoort aan het opgegeven e-mailadres. Indien dit Account-object niet bestaat, geef je null.

Test beide methoden met unit testen.



c	AccountRepository	
f	accounts	
m	addAccount(Account)	void
m	findAccount(String)	Account

Figuur 5: Klasse AccountRepository

## 6 De toepassing uitvoeren

---

Alle functionaliteit is nu aanwezig om de toepassing uit te voeren. Om de applicatie te starten, run je de main-methode in de klasse **StreamingServiceAppStarter**.

Eerst zal je een account moeten aanmaken. We gaan ervan uit dat de gebruiker alle informatie netjes invult. Vanaf volgend hoofdstuk gaan we onze toepassing robuuster maken en zorgen dat een foutieve ingave netjes wordt afgehandeld.

Nadat de account is geregistreerd kan de gebruiker inloggen. Afhankelijk van de geboortedatum die je invult voor het kijkersprofiel (door op de profiel-image te klikken) krijg je aangepaste content te zien.

Als alles correct werkt ben je klaar om te starten met het volgende hoofdstuk!

