Project Kantine

1.4

Auteur: Teun de Jong, Stijn Wolthuis

Docent: Harald Rietdijk

Datum: 18-05-2020

## Week 1

### 1B. Leg uit wat de begrippen declaratie en initialisatie betekenen.

Declaratie: Het aanmaken van een variabele, zo weet de compiler dat we een variabele gaan gebruiken met een bepaalde naam en van welk type.

Initialisatie: Voor de eerste keer een waarde toekennen aan een variabele.

## Week 2

### 2A. Leg uit waarom het gebruik van een while lus in de methode verwerkRijVoorKassa() handiger is dan een for lus.

Er is minder code nodig om hetzelfde resultaat te bereiken. Werkt met Boolean, je hoeft geen extra teller bij te houden.

Met een for lus worden er personen overgeslagen. Door de teller

### 4B. Bij welke methodes in Kassa en Kantine komt dit voor(dubbele methode).

HoeveelheidGeldInKassa() en AantalArtikelen().

### 5A.Leg uit waarom het goed is om de methodes ArrayList<Artikel>getArrayList(Stringproductnaam)enArtikelgetArtikel(ArrayList<Artikel>)privatete maken.

Omdat het belangrijk is dat niet iedereen deze methode kan aanroepen.

### 5B.In welke situatie gebruik je een HashMap en wanneer een HashSet?

Hasmap is voor het opslaan als iets 2x in de map voor mag komen.

Hashset als het uniek moet zijn en een waarde niet dubbel mag zijn.

### 6A. Leg de werking van de constructor uit.

De constructor maakt een nieuwe kantine aan en een random getal. Dan een Array int[] hoeveelheden krijgt de waarden van getRandomArray(aantal\_artikelen, min\_artikelen\_per\_soort, max\_artikelen\_per\_soort). Vervolgens wordt er een nieuwe kantineaanbod aangemaakt en deze wordt gezet met het huidige kantineaanbod.

### 6B. Leg de implementatie van intgetRandomValue(int min, int max) uit en met name waarom er +1 in voorkomt.

De methode genereert een waarde binnen de opgegeven parameters. De reden waarom +1 wordt gebruikt is omdat je anders niet aan je minimale en maximale waarden kan komen.

## Week 3

### 2C. Er is geen constructor gedefinieerd voor Administratie terwijl je gewoon new Administratie() kan aanroepen. Leg uit waarom dat kan

Er wordt automatisch door Java een lege constructor gemaakt als deze niet word aan gemaakt.

### 2D. Leg uit waarom de twee al bestaande methoden van Administratie static kunnen zijn.

Deze methode kan static zijn omdat we deze simpele methoden voor het uitrekenen van het gemiddelde niet afhankelijk hoeven te zijn van een gemaakt Administratie object.

We kunnen het overal aanroepen en een static methode kunnen we aanroepen op basis van de klasse en niet het gemaakte object.

### 2E. Als je dat wil voorkomen kun je een private constructor voor Administratie maken. Doe dat en leg uit waarom je je doel nu bereikt.

Er veranderd niets omdat hij eerst zelf gemaakt werd en nu door ons.

### 2G.Leg uit wat final doet.

Een final waarde, methode of klasse kan niet worden veranderd. Nadat deze is aangemaakt kan deze niet worden gewijzigd.

### 2H. Als het goed is klaagt de compiler over zoiets als “Cannot make a static reference to the non-static field ...”. Leg uit waarom de compiler hierover klaagt.

De compiler klaagt over het maken van een static reference naar een non-static field omdat een static methode geldt voor de gehele klasse zonder een instantieobject. Omdat een non-static field alleen komt te bestaan bij het instantiëren van een object kan een static methode niet refereren naar een non-statisch veld omdat deze niet altijd hoeft te bestaan.

### 2I. Welk “probleem” heb je nu geïntroduceerd?

Nu staat het veld DAYS\_IN\_WEEK niet vast en zou het later opnieuw gedefinieerd kunnen worden en dat zou exact zijn wat we niet willen.

### 3B. Waarom moet een super aanroep in de constructor altijd bovenaan staan?

Zorgt ervoor dat een parant klasse goed word geïnitialiseerd zodat deze goed kan worden aangeroepen.

## Week 4

### 1B. Teken een sequentie-diagram

### 

### 1C. Waarom is de instantie variabele saldo protected gemaakt? Waarom is dat handig?

Niet iedereen mag het saldo van de persoon zien, behalve de persoon zelf. Net zoals in het echt. Omdat deze anders niet benaderbaar is.

### 3A. Kun je een instantie maken van een interface via new? Leg uit waarom het logisch is dat het wel of niet kan.

Nee, dit kan niet. Een interface mag wel als datatype dienen, maar je kunt er geen instanties van maken. Alle methodes in de interface hebben geen body , dus een instantie van een interface zou geen waarde hebben.

### 3B. Herhaal de vorige vraag met abstracte klassen.

Van abstracte klassen kunnen ook geen directe instanties gemaakt worden. In een abstracte is geen complete implementatie van de klasse, maar een basis voor subklassen.

### 3C. Kan een klasse meerdere klassen overerven?

Nee, een (abstracte) klasse kan maar een keer overerven

### 3D. Kan een klasse meerdere interfaces implementeren?

Ja, een klasse kan meerdere interfaces implementeren.

### 3E. Kan een klasse tegelijk een klasse overerven en interfaces implementeren?

Ja. Wel schrijf je extends voor implements en niet andersom.

### 3F.Klopt de stelling dat elke methode in een interface abstract is? Licht je antwoord toe.

Een interface is een puur abstracte klasse. De klasse en al zijn methodes zijn abstract.

### 3G.Moet een klasse abstract zijn als minstens één methode abstract is? Licht je antwoord toe.

Ja, je kunt geen abstracte methoden schrijven in een niet abstracte klasse. Andersom is niet waar. Een abstracte klasse hoeft geen abstracte methoden te bevatten. Het is immers geen complete implementatie van de klasse.

### 3H.Leg het begrip polymorfisme van klassen uit en geef twee voorbeelden (één met abstracte klas-sen en één met interfaces).

polymorfisme is de eigenschap van een object om verschillende vormen aan te nemen. Bij een interface kan dit betekenen dan een koe (klasse) een dier (interface) is en zelf een object (klasse) is. Koe k is een koe. Koe k is een dier. Koe k is een object. Bij abstracte klassen werkt het vergelijkbaar. Stel je hebt een koe klasse die je abstract maakt. Dan kun je geen directe instanties meer maken van koe. Maar onder koe staat een actieveKoe klasse die koe extend en een luieKoe klassse die koe extend. Dan kun je wél een instantie van actieveKoe of luieKoe maken.

### 4A. Kan een klasse abstract zijn als geen enkele methode abstract is in die klasse? Probeer het eens uit. Leg waarom het logisch is dat dit wel of niet kan.

Ja, je kunt wel een abstracte klasse zonder methoden hebben, maar geen abstracte methoden zonder abstracte klasse. Je kunt immers geen instanties maken van abstracte klassen.

### 4B. Moet een subklasse van een abstracte klasse altijd alle abstracte methodes implementeren? Leg uit waarom het logisch is dat dit wel of niet kan.

Ja, dat moet wel. Als je dit niet doet, moet de subklasse ook abstract gemaakt worden. Wanneer je dit niet zou doen, zouden er lege methodes zijn.

### 4C. Als een klasse niet alle methoden van een interface implementeert kun je iets doen om een(compiler)fout te voorkomen. Wat? Waarom is de oplossing logisch?

Het doel van een interface, is om ervoor te zorgen dat alle subklassen van deze interface de methoden uit de interface implementeert. Als je de methode écht niet wilt gebruiken, kun je de methode null laten returnen of een NotImplementedException gebruiken.

### 4D.Leg uit waarom het logisch is dat een instantie variabele niet abstract kan zijn.

Een instantie variabele kan niet abstract zijn, omdat deze concreet is.

### 4E.(Uitdaging) Zoek uit wat een final methode is. Leg daarna uit waarom het logisch is dat een methode niet tegelijkertijd abstract en final kan zijn.

Een abstracte methode is incompleet. Een final methode is compleet. Ze spreken elkaar dus tegen en kunnen niet tegelijk in 1 methode gebruikt worden. Immers moet een abstracte methode ergens anders extend worden, en een final methode kan dit niet.

## Week 5

1D. Lees de klassen Student ,StudieInschrijvingen Telefoondoor. Zou je al iets kunnen ver-tellen over het doel van de annotaties @Id, @GeneratedValueen @Column?

id is primary key

column is een nieuwe kolom in de tabel

Generated value geeft aan dat de waarde automatisch wordt geïncrementeerd.

### 1E. Een vier tal tabellen zullen nu in de database zijn aangemaakt, welke zijn deze?

Student

Student\_telefoon

Studie

Telefoon

### 2A. Het attribuut telefoons in de klasse Student is geannoteerd met @OneToMany. Waarom zou hier @OneToMany en niet @OneToOne zijn gebruikt?

Een student kan me erdere telefoons hebben

### 2B. Hoe wordt het verschil zichtbaar in de database?

In de tabel studie, kunnen studenten meerdere keren voorkomen. In het gegeven voorbeeld zien we dat student\_id 11 meerdere keren voorkomt. Echter kan in de tabel student, een student maar een enkele keer voorkomen.

### Zou je een nadeel kunnen bedenken waarom voor de relatie Studenten StudieInschrijving via studies geen @JoinTable is gebruikt?

Jointable geeft minder redundancy dan joincolumn, bovendien wordt het er overzichtelijker van.

### 2C. Beschrijf hoe deze relatie in de database zichtbaar wordt.

Meerdere studenten kunnen dezelfde studie hebben.

### 2D.Waar vind je eenStudentKaartterug in de database?

De studentkaart kun je terug vinden in de tabel Student. Omdat deze embedded is, zien we in de tabel kaartNummer en vervalDatum. De eigenschappen van de kaart.

### Wat zou je hieruit kunnen afleiden met betrekking tot het gebruik en de functie van @Embeddable en @Embedded?

Embeddable betekent dat de eigenschappen ergens anders in gaan. In dit geval Student. Bij student staat dan ook embedded private StudentKaart.

3A.Een transactie wordt ook wel een unit of atomicity genoemd, waarbij atomiciteit in de context van databases „alles of niets” betekent. Vooruitkijkend op week 6, kan je voor de kantinesimulatie een situatie bedenken waar je dit patroon zou kunnen toepassen?

Wanneer een betaling bij de kassa niet gelukt is, moet de informatie niet in de database terecht komen. De query moet stoppen en pas weer opnieuw gestart worden, bij een nieuwe betaling.