

Inhoudsopgave: Datum: 26/09/2023

[1 Soorten types 4](#_Toc155338849)

[2 Hoofdstuk 3 (Domeinklasse) 5](#_Toc155338850)

[2.1 Object 5](#_Toc155338851)

[2.2 Abstractie 5](#_Toc155338852)

[2.3 Klasse 5](#_Toc155338853)

[2.4 Opbouw van een klasse 5](#_Toc155338854)

[2.5 Opbouw van een klasse – met objecten 5](#_Toc155338855)

[3 Hoofdstuk 4: Arrays en ArrayLists 7](#_Toc155338856)

[3.1 Array 7](#_Toc155338857)

[3.2 ArrayList 8](#_Toc155338858)

[3.3 Tweedimensionale arrays 8](#_Toc155338859)

[4 Hoofdstuk 5: Ontwikkel een applicatie 10](#_Toc155338860)

[4.1 Theorie 10](#_Toc155338861)

[4.2 Visual Paradigm 13](#_Toc155338862)

[5 Hoofdstuk 6 Methodes Next Level 15](#_Toc155338863)

[5.1 DTO 16](#_Toc155338864)

[5.2 Enums 17](#_Toc155338865)

[6 Hoofdstuk 7: pijlers van de OO 18](#_Toc155338866)

[6.1 4 pijlers 18](#_Toc155338867)

[6.2 4 pijlers uitgewerkt 18](#_Toc155338868)

[7 Hoofdstuk 8: Testen 21](#_Toc155338869)

[7.1 We love testing 21](#_Toc155338870)

[7.2 Ontwerp technieken 21](#_Toc155338871)

Exameninfo

Geen theoretisch examen, pure praktijk! Openboek examens/testen!

Halverwege semester 1, één grote test/toets met de 1e 3 hoofdstukken! 25% van de punten voor dit vak. Max punten wat je kan halen is 5/20 dankzij dit. BLOKKEEEEEEEEEEEEEEEEEEEEEEEEEEEN!

Donderdag 9 november 10:30 is de datum van de midtermtest. Tijd = 1.5u

Modulo operator is heel belangrijk voor de midtermtest.

Voor de midtermtest KRIJG je de UML. Op het examen moet je die wel kunnen maken.

# Soorten types

|  |  |
| --- | --- |
| Type | Uitleg |
| int | Variabele voor een getal |
| void | Het maken van een lege methode, je verwacht niks terug. |
| Double | Variabele voor een kommagetal |
| String | Variabele voor een stukje tekst |
| Boolean | Variabele met als uitkomst true of false |
| LocalDate | Type voor het maken van een datum |
| BigDecimal | Specifiek type voor het berekenen van geld |
| SecureRandom | Variabele voor het maken van een random generated getal. |
| DTO | Type voor een lijst te maken met een methode van verschillende types. |
| RepositoryType | Dit type is enkel bij een repository en krijgt de exacte naam van de parameter/attribuut in kwestie. Dit is namelijk de naam van het returntype. |
| long | Een int, maar gemaakt voor grotere getallen |

# Hoofdstuk 3 (Domeinklasse)

## Object

Definitie object:

* Een **real-life object** zoals een pennenzak.
* Een (real-life) object heeft ook **eigenschappen** zoals de pennenzak die
  + Kleur, lengte, hoogte, breedte en materiaal heeft.

gedrag:

Dit omdat wat je kan doen met het object (**methoden**).

## Abstractie

Definitie abstractie:

het enkel opnemen van de **noodzakelijke** eigenschappen en/of gedrag en tegelijk het verbergen van onnodige details.

## Klasse

## Opbouw van een klasse

|  |  |
| --- | --- |
| Stap | Uitleg |
| 1 | Maak een classe in de package ‘**domein’** ZONDER public static void main! |
| 2 | maak je **eigenschappen** (private) |
| 3 | maak van de eigenschappen **getters** en **setters** (<-> + public) |
| 4 | Maak de **constructors** (Tip herkennen: hebben de naam van de klasse) ipv this.naam gebruiker de setters (setNaam(naam)) zodat controles ook in de constructors gebeuren. |
| 5 | Maak de functies handmatig aan met **parameters**. |
| 6 | Test je code uit aan de hand van een **Unittester**! Maak de JUNIT aan:   * Rechtermuisknop project * Build path * Add Libraries * JUNIT |
| 7 | Setters en functies aanpassen aan de hand van de **domeinregels**   * Herhaal geleidelijk de JUNIT tot als alle fouten weg zijn. |

## Opbouw van een klasse – met objecten

|  |  |
| --- | --- |
| Stap | Uitleg |
| 1 | Een **nieuw product** maken:   * Product (naam class) pannenkoek (naam object) = new Product(“Pannenoek”, 0.125, 21, 5) (naam class + invulling eigenschappen) |
| 2 | Indien je **data** vraagt aan de gebruiker maak dan een **aparte klasse** waarin je vraagt wat de gebruiker wil: |
| 3 | Berekeningen maken indien nodig |
| 4 | Programma **robust** maken door **validatie** toe te voegen. |

# Hoofdstuk 4: Arrays en ArrayLists

## Array

Definitie array:

Een array is een lijst met een **vaste** waarde.

Opbouw Array:

* We weten hoe een normale int wordt opgesteld (int getal = 5) Hier is het verschil dat dit
  + int[] getal = new int[]
  + Het krijgt rechte haakjes en wordt geïnitialisseerd met het type van de array.

(Zero based) Index:

Afbeelding met tekst, Lettertype, lijn, nummer

Automatisch gegenereerde beschrijvingHet decimale getalsysteem werkt met: getal 1 = 1, getal 2 = 2 … met arrays is dit het decimale systeem – 1. Dus getal 0 = 1, getal 1 = 2 …

* Het feit dat een array start met index[0] noemt men **zero based indexing**.

Een getal van een array weergeven:

Een getal van een lijst geven we weer door %d en **naam\_lijst[index]** te typen.

Array een waarde geven na het maken van een lijst:

We typen **naam\_lijst[index] = waarde** die jij wil.

We kunnen ook typen **naam\_lijst[index] = “waarde” + de indexwaarde van een vorige index.**

Array.length():

Om de grootte van een lijst weer te geven typen we **%d met naam\_lijst.length().**

SecureRandom sr = new SecureRandom();

Zo maak je een **random** getal. Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

Als de randomwaarde ‘waar’ is krijg je deze lijst van 7 getallen, zoniet één van 3 getallen.

Klassieke for-lus en enhanced for-lus:

**Klassiek:**

* Afbeelding met tekst, Lettertype, schermopname, lijn

  Automatisch gegenereerde beschrijving
* Hierbij overlopen we elke index van de lijst en tellen we het op bij een som.
* **Verschil**: hier heb je **meer** **controle** over wat je doet.
* Hier moet je werken met **getallen[index]**

**Enhanced:**

* Afbeelding met tekst, Lettertype, handschrift, wit

  Automatisch gegenereerde beschrijving
* Hierbij overlopen we elke index van de lijst en tellen we het op bij een som.
* **Verschil**: Hier is de code tussen de {} makkelijker om te schrijven wat werkt onder het KISS-principe **(Keep It Short and Simple).**
* Hier moet je werken met **getal (bepaalt door de enhanced for-lus)**

## ArrayList

Definitie ArrayList:

Een ArrayList heeft een **variabele waarde wat betekent dat de grootte dynamisch is.**

Opbouw ArrayList:

ArrayList**<type>** naam = new ArrayList<>() ;

Hier zijn de types anders gedefinieerd, daarom gebruiken we **wrapper-classes**.

Wrapper-classes (te gebruiken bij ArrayLists):

|  |  |
| --- | --- |
| int | Integer |
| double | Double |
| char | Character |
| boolean | Boolean |
| float | Float |
| byte | Byte |
| short | Short |
| long | Long |

Elementen toevoegen - ArrayList.add():

Hiermee **voegen** we een **waarde** **toe** aan de lijst.

Aantal elementen - ArrayList.size():

hiermee tonen we de **grootte** van de ArrayList.

Aantal elementen - ArrayList.isEmpty():

Hiermee kijken we als de ArrayList **leeg** is.

Elementen ophalen – ArrayList.get(index):

Hiermee **halen** we de **waarde** van de ArrayList via een index **op**.

Elementen wijzigen – ArrayList.set(index):

Hiermee **wijzigen** we een **element** in de lijst.

Elementen verwijderen – ArrayList.remove(index):

hiermee **verwijderen** we een **element** in de lijst.

## Tweedimensionale arrays

Definitie:

Een tweedimensionale array is een **lijst** gewerkt **in** **een** **lijst** [**matrix**] (**geneste** **array**).

* De eerste array staat voor *aantal rijen*.
* Afbeelding met diagram, lijn, schermopname, nummer

  Automatisch gegenereerde beschrijvingDe tweede array staat voor *aantal kolommen*.

Voorbeeld tweedimensionale array:

int[][] matrix = new int[3][4]

Ook al staat er *3* en *4*, ze werken alsnog met **indexen**, dus:

0, 1, 2 in **kolommen**.

0, 1, 2, 3 in **rijen**.

Waarden toevoegen in een matrix:

Je roept de naam van de matrix op **(matrix[getal(kolom)][getal(rij)])** en vult op de plek van de ***getal*** je **waarde** in.

Waarde invullen voor een volledige rij:

Afbeelding met tekst, Lettertype, schermopname, lijn

Automatisch gegenereerde beschrijving

matrix overlopen:

Een matrix overlopen gebeurt met 2 verschillende (enhanced) forlussen.

Afbeelding met diagram, lijn, schermopname, tekst

Automatisch gegenereerde beschrijvingJagged Arrays:

Soms kan een matrix ook **verschillende** kolommen en rijen bevatten, dit is een **jagged array.**

* Voorbeeld: int[][] jagged = int { {11, 22, 33} / {-2, -5} / {4, 4, 4, 4} }
  + (**3** **kolommen** en **3/2/4 rijen**).

# Hoofdstuk 5: Ontwikkel een applicatie

## Theorie

**Visual Paradigm komt op het examen!**

Wat is een associatie?

Een relatie tussen twee verschillende klassen.

Wat zijn multipliciteiten?

Het aantal eenheden op je associatie {0, 1, x, \*)

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

|  |  |
| --- | --- |
|  |  |
|  | Compositie (zwart ruitje) |
|  | Navigeerbaarheid  Je bepaalt in welke richting de associatie zich mag wijzen. Je volgt die maar in een bepaalde richting. |
| 1e ? | Multipliciteiten |
| 2e ? | Visibiliteit (private, public …) |
|  | Naam eigenschap |

Strikvraag: hoeveel eigenschappen heeft Aquarium?

4 eigenschappen:

3 eigenschappen in AquariumWinkel

Aquarium telt als 1 eigenschap voor AquariumWinkel. Dit is de nieuwe eigenschap.

Afbeelding met tekst, schermopname, Lettertype, diagram

Automatisch gegenereerde beschrijving

Bij multipliciteiten vraag je je alleen maar af hoeveel de verbonden klasse heeft in multilpiciteiten.

Wat is een knowing verantwoordelijkheid?

Een object heeft zijn verantwoordelijk, deze methode moet iets weten.

Wat is een doing verantwoordelijkheid?

Een object heeft de verantwoordelijkheid iets te doen.

Wat is het DCD?

Hetgene wat je creeërt in Visual Paradigm.

Eigenschappen controller patroon:

Naam: Controller

Probleem: Aan welk object gaan we vragen dat verantwoordelijk is voor het ontvangen en coördineren van een systeemoperatie.

Oplossing: kan de verantwoordelijkheid toe aan een klasse die het volledige systeem

Samenvatting: Alles wat moet gebeuren met de **UI**, **communiceert** met de **domeincontroller**.

Eigenschappen creator patroon:

Naam: Creator

Probleem: Wat is er verantwoordelijk voor het aanmaken van nieuwe instanties van een klasse?

Oplossing: Ken de verantwoordelijkheid om instanties van klasse A te creëren toe aan klasse B, wanneer 1 of meer van de volgende redenen gelden:

* + - B is een aggregatie of een compositie van A
    - **B bevat instanties van A (eigenschap: bevat, heeft)**
    - B gebruikt A intensief
    - B bevat de data om A te initialiseren

Eigenschappen expert patroon (wie heeft de informatie die het antwoord voor je heeft):

**Naam:** Expert

**Probleem:** Wat is het algemeen principe dat we kunnen hanteren voor het toekennen van **verantwoordelijkheden** aan objecten?

**Oplossing**: Ken de verantwoordelijkheid toe aan de Information Expert – dit is de klasse die alle informatie, nodig om de verantwoordelijkheid te realiseren, heeft..

Hoe starten we aan een nieuw project?

* 1: We kijken naar het SSD dat de klant heeft meegegeven en nemen (bijna) alle operaties over in een **domeincontroller**.
* Doorgaande pijl (🡪 met Operation Contract) = **VOID** methode.
* Doorgaan pijl (🡪 zonder OC) = is er een tussentoestand nodig waarbij iets word bijgehouden? Ja = **VOID**  Nee = de terugkeerpijl meenemen in de methode
* Terugkeerpijl (pijl met stippellijntjes) zijn meestal een geefMethode.
* 2: Voor de **klassen** kijken we naar het **domeinmodel**.
* We nemen de associatie volledig over, maar de multipliciteiten nog niet.
* 3: alle **methodes** uit de **domeincontroller** uitwerken.
* Nu laat je ze werken via het **creator patern** en het **expert patern**.
* We creëren bij beide klassen een constructor.
* Indien je in het OC “**gecreërde / gemaakt**” ziet, volg dan het **creator patern**. Zie je dit niet? Dan moet je het **expert patern** werken.
* We tekenen een **associatie** tussen domeincontroller en de klasse waarvan hij een object moet maken. De kant langs de **controller** is **ALTIJD FALSE**.
* Kom je een aggregatie of compositie tegen, dan is het **geheel verantwoordelijk** voor het **aanmaken van de onderdelen**.

RDD (domeincontroller – Responsibility Driven Design)

Alle operaties van de use case gaan in de domeincontroller, hij is namelijk verantwoordelijk voor de UI.

Hoe transformeren we code naar Eclipse?

In Eclipse: rechtermuisknop op src map, ga naar ‘show in’. Kopieer het pad.

Visual Paradigm: ga naar Tools, Code Engineering, en genereer code.

Random Number Generator:

private SecureRandom sr;

Een goede softwareapplicatie bestaat meestal uit 3 belangrijke lagen:

domeinlaag – databanklaag –

Repository:

* Een soort **databanklaag** die zich verbindt met een klasse waarbij gegevens moeten worden bijgehouden voor een lange tijd. Hij staat in voor het beheer van een groep objecten.
* Een repository kan **CRUD** operaties uitvoeren: **Create,** **Read, Delete en Update**.
* Je verbind ALTIJD **van** de **repository** **NAAR** de **klasse** in **kwestie**.
* Je verbindt ook ALTIJD **van** de **domeincontroller** **naar** de **repository**
* Als je een **speler** toevoegd heeft dit type **Speler**
* Naam is altijd: **naamRepository** (dit staat altijd achteraan van het woord)
* Wil je een **lijst van alle objecten**? Maak dan een **geefMethode** aan in de repository
* Het **type** van de repository is een **return** type.

Skelet repository:

Afbeelding met tekst, schermopname, lijn, diagram

Automatisch gegenereerde beschrijving

DTO-object:

* Data Transfer Object
* Een lijst van gemixte types.
* Doel = data transfereren van de domeinlaag naar de UI-laag.
* Dit wordt enkel gebruikt als er verschillende types zijn in de paramters van de domeincontroller.
* Naam =

ArrayList:

Afbeelding met tekst, Lettertype, schermopname, lijn

Automatisch gegenereerde beschrijving

Extra uitleg selectie:

Afbeelding met tekst, schermopname, lijn, Lettertype

Automatisch gegenereerde beschrijving

Bij een geselecteerde … heb je altijd een getter nodig bij hetgeen wat je nodig hebt (deze attribuut is te vinden bij de domeincontroller)

## Visual Paradigm

Hoe maak je een nieuw project aan?

File 🡪 New Project 🡪 naam invullen

Rechtermuisknop op het bestand dat je wil maken en nieuw klasse aanmaken.

Hoe maak je een nieuw domein aan?

Hoe maak je in een domein klassen aan, met methodes, attributen … ?

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

Rechtermuisknop ergens in de klasse {add operations}.

Hoe passen we methodes hun toegankelijkheid aan?

Afbeelding met tekst, software, lijn, Perceel

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, visitekaartje, lijn

Automatisch gegenereerde beschrijving

**Eenmaal** de **UI is gemaakt** via de DomeinController spelen we dit door naar de **klasse** die het **nodig** **heeft** en hij erft dan deze UI over. Spel kan **NOOIT** verbinden naar DomeinController. Altijd van controller naar de klasse.

De bedoeling van van de Domeincontroller is dat je methode per methode afgaat om ze te programmeren.

Een klasse is altijd verantwoordelijk voor al zijn attributen (elke avond 10x afspelen).

# Hoofdstuk 6 Methodes Next Level

Method/constructor overloading:

Methodes/constructors kunnen dezelfde namen hebben, maar MOETEN verschillende **parameters** hebben.

Paramters:

Een variabele die je doorgeeft in de haakjes van een methode, je kan die enkel gebruiken binnenin die methode.

Modifiers:

* Acces modifier (toegang)
  + **Public**
  + **Private**
  + **Static**

(het bestaat maar één keer, enkel in het niveau van de klasse – het zit niet bij de eigenschappen van het object – alle **objecten** krijgen **dezelfde** **waarde**)

* + **final** (geef je die een bepaalde waarde? Dan kan je die niet meer veranderen)

Finals in constructors:

* Attributen die **geen** **final of static** zijn = this.setLengte(lengte);
* Attributen die **final of static** zijn = this.lengte = lengte;

Argument:

De waarde van de parameters die worden doorgegeven.

Stack:

* **Push**: Er wordt iets op de stapel gelegd.
* **Pop**: Er wordt iets van de stapel gehaald
* Dit principe werkt via LIFO (Last In First Out)

Pass by value:

Eclipse werkt met deze techniek wat betekent dat ingegeven waarden (value/waarde) altijd worden doorgegeven. (de declaratie van een variabel wordt dus ongewijzigd)

Trow exception:

* Throw new IllegalArgumentException(“”);
* This.attribuut = attribuut
* Dankzij deze error gebeurt validatie vanaf nu in de methodes.
* Eenmaal een error naar boven komt stopt de normale verloop van het programma en keert het programma met al zijn stappen terug.

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

Deze klasse heeft 2 eigenschappen (static telt niet mee).

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

Static en final in UML:

* Static in UML staat onderlijnt – Dit zijn instantie-/classvariabelen
* Final wordt zo weergegeven **{final}**
* Is een attribuut final en static? dan wordt het in hoofdletters geschreven (constanten)
* Het is gebruikt om methodes makkelijker te maken

Static en final in Eclipse:

* Worden in hoofdletters geschreven
* Krijgen altijd een waarde bij het maken van de attributen.

String object leeg in een methode:

if (attribuut == null || attribuut.isBlank())

Een attribuut verhogen in de constructor:

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

als een bepaald attribuut verhoogd, stel je dit in de hoofdconstructor en dat wordt dan automatisch doorgegeven in de andere constructors.

Verklaring DomeinController:

Afbeelding met tekst, Lettertype, schermopname, lijn

Automatisch gegenereerde beschrijving

Doos = naam van het domein

dozen = naam uit het UML-diagram van de associatie

## DTO

Record klasse:

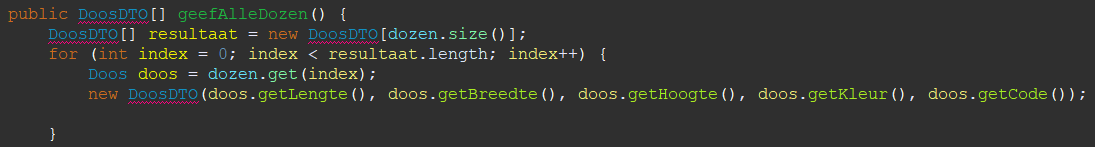
Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Bij het maken van **DTO-klasse**, maken we een nieuw **RECORD** aan IPV een CLASS.

DTO is 5 punten van de 75 op het examen.

DTO-objecten overlopen:



Zo maak je een lijst van een DTO waarbij je elk objectje overloopt.

## Enums

Enum:

Hier zet je enkele **vaste waarden** in uitgedrukt in constanten (final en static).

Ook hier mag je NOOIT een **publieke constructor** hebben, enkel maar een **private**.

Enum omzetten naar een string in CUI:

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Gordel.values toont alle waarden in deze Enumklasse.

Bij cases mag je enkel maar de Enum zelf typen. Niet de klasse met een puntje daarvoor.

Stack Overflow error:

Eén zelfde error die zich steeds voortdoet honderden keren tot de stack vol zit en geen errors meer kan tonen.

# Hoofdstuk 7: pijlers van de OO

## 4 pijlers

4 pijlers van de OO:

* **Inkapseling Encapsulation**
  + Zorgen dat objecten in een reële toestand bevinden
* **Abstractie Abstraction**
  + Je neemt enkel de gegevens op die je nodig hebt van objecten.
* **Overerving Inheritance**
  + Een systeem om herhaling te vermijden.
* **Polymorfisme** **Polymorphism**
  + Een object kan verschillende klassen zijn (soms een hond of een kat …)

Open pijlen:

Open pijlen in het domeinmodel bij overerving betekenen “IS EEN”.

xxx.equals:

Dit gebruik je om Strings met elkaar te vergelijken of ze dezelfde waarde/naam hebben.

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

Volgorde objecten maken en functies oproepen:

Het maakt niet uit waar je een nieuw object maakt en oproept.

Aanroepen van overerving:



Super() in een constructor:

Deze functie roept de constructor van de superklasse aan.

Afbeelding met tekst, Lettertype, handschrift, schermopname

Automatisch gegenereerde beschrijving

## 4 pijlers uitgewerkt

Methodes overriden:

* De linkse afbeelding is de superklasse
* De rechter afbeelding is de subklasse
* Indien je dezelfde methode van de superklasse override in de subklasse wordt die waarde van de methode overgenomen.

Afbeelding met tekst, Lettertype, schermopname, ontwerp

Automatisch gegenereerde beschrijvingAfbeelding met tekst, Lettertype, schermopname, Graphics

Automatisch gegenereerde beschrijving

@override en supermethodes oproepen:

* Met **@override** maak je duidelijk in de subklasse zijn domeinklasse dat de waarde van die methode is overgenomen.
* Je kan ook de exacte methode oproepen van de superklasse door het keyword **super. toe** te voegen aan de methode.

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Moederklasse Object:

Elk object heeft een moederklasse namelijk Object, in deze klasse zijn enkele functies:

Belangrijk voorbeeld polymorfisme:

Afbeelding met tekst, schermopname, Lettertype, software

Automatisch gegenereerde beschrijving

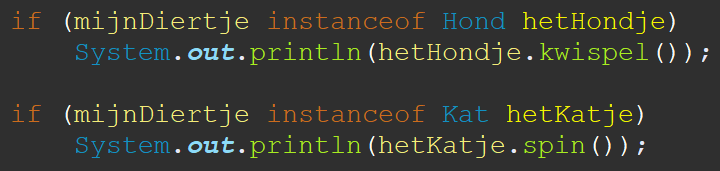
Downcast:

We forceren een object van de superklasse om naar de subklasse.



instanceof:

Indien een object van een superklasse overeen komt met een object van de subklasse, dan …



Protected:

Nu mag je dit overal gebruiken in de PACKAGE.

Afbeelding met tekst, Lettertype, schermopname, ontwerp

Automatisch gegenereerde beschrijving

Controleermethode:

Hier dwing je validatie af, maar je geeft het hier geen waarde. Dit is voor final/static attributen.

Subklasses maken met direct superklasses en constructors:

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijvingAfbeelding met schermopname, tekst, Multimediasoftware, software

Automatisch gegenereerde beschrijving

Overerving bij subklasses in constructors:

Afbeelding met schermopname

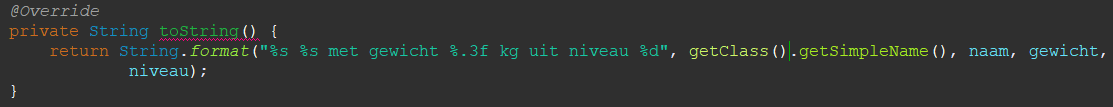
Automatisch gegenereerde beschrijving

Nieuwe attributen bij een subklasse krijgen in de constructors ook die attributen.

Nooit overrideble methods aanroepen vanuit de constructors:

* ofwel roep je private methodes aan
* ofwel roep je public (set)methodes aan die **final** zijn.

toString (naam klasse oproepen – getclass().getSimpleName()):



persistentielaag:

Dit is een databanklaag.

ORM:

* Object Relaional Mapping
* Je verzoent de objectwereld met de echte wereld.
* Dit gebeurt meestal met tools, maar voor kleinere objecten moet je dit best zelf doen.

Objecten forceren in een subtype:



Het wapen is een **downcast**.

# Hoofdstuk 8: Testen

## Afbeelding met tekst, schermopname, cirkel Automatisch gegenereerde beschrijvingWe love testing 😉

Unit testen:

Unit testen is een vorm van software testen waarbij er afzonderlijke eenheden (units), i.e. kleine stukjes logica, onafhankelijk van elkaar, snel en grondig worden getest. Het doel is te valideren dat de units exact doen wat er van hen verwacht wordt. Bij object georiënteerde software ontwikkeling is een typische unit een methode in een klasse.

Afbeelding met tekst, Lettertype, logo, Graphics

Automatisch gegenereerde beschrijvingTest driven Development:

* Je gaat je unit testen schrijven voor je begint met programmeren van de applicatie.
* Hier draai je de hele tijd rond in deze cirkel.
  + Eerst gaat iets meestal falen
  + Daarna werkt het
  + Dan kijk je eventueel of je iets kan Refactoren.

FIRST eigenschappen:

* **Fast:**
  + Unit Testen MOETEN snel runnen.
* **Isolated**:
  + Je gaat meestal niet heel breed testen, bij een gefaalde test ga je dan direct naar de methode die faalt.
  + Ook hebben ze nooit order-of-run afhankelijkheid van elkaar. Je zou elke test op zichzelf kunnen uittesten zonder dat een andere test eerst moet gebeuren.
* **Repeateble:**
  + Je moet wanneer dan ook deze test kunnen uitvoeren en hij moet altijd hetzelfde resultaat geven.
    - Hangen niet van externe services (network, database)
    - Opletten met gebruik van LocalDate.
* **Self Validating:**
  + Geen manuele check nodig om te weten of een test slaagt of faalt, het wordt automatisch aan jou verteld.
* **Timeley:**
  + De testen maken mogen niet te lang duren
* Het moet geschreven zijn voor er wordt gecodeerd

## Ontwerp technieken

Test cases:

* **Positief testen:** 
  + test cases die **geldige** data aanleveren
  + Kijk of de code **effectief** **doet** wat ze **verondersteld** wordt.
* **Negatief testen:** 
  + Test cases die **ongeldige** data aanleveren
  + Kijken of de code **effectief** reageert wanneer **ongeldige data** wordt aangeleverd.

Equivalentiepartities (EP):

* Een interval van gegevens waarvoor je eenzelfde gedrag van een methode verwacht.
  + Deze techniek kan je gebruiken wanneer in de domeinregels bereiken, intervallen worden vermeld.
  + Om testgevallen te bepalen gaan we ons beperken tot 1 willekeurig data element per partitie.
  + Het getal moet bv. positief zijn, maar ongeacht welke negatieve prijs, krijg je altijd dezelfde feedback (1 negatief testgetal is dus goed genoeg)

Geldige EP:

Dit is met normaal gedrag. Ze leiden tot positieve testen.

Negatieve EP:

Dit abnormaal gedrag. Hierbij wordt een Exception geworpen. Ze leiden tot negatieve testen.

Tip UnitTesten maken:

* Je neemt een paramater en maakt op een getallenas bij ints en doubles.
  + Het interval dat geldig is kleur je groen.
  + Het interval dat ongeldig is kleur je rood.
  + Afbeelding met schermopname, lijn, Rechthoek

    Automatisch gegenereerde beschrijving
  + De grenzen eindigen 1 – positief en 1 + positief van de getallen as.
  + Hierbij heb je enkele unittesten:
    - -5 = Exception (Random getal buiten de boundry values)
    - 20 = Exception (Random getal buiten de boundry values)
    - 5 = Bestelling gemaakt (Random getal tussenin de boundry values)
    - 0 = Ecxeption (-1 van de boundy value)
    - 11 = Exception (+1 van de boundry value)
    - 1 = Bestelling gemaakt (Boundry values – kleinst toegelaten aantal)
    - 10 = Bestelling gemaakt (Boundry values – grootst toegelaten aantal)
    - …
* Bij booleans zal de waarde altijd geldig zijn, MAAR het zal TRUE of FALSE zijn.

Stappenplan testen maken Equivalentiepartities:

* Willekeurige waarden kiezen na en voor de geldige boundrys (2 partitites)
* Kijken of de boundry(‘s) geldig of ongeldig zijn.
* De boundrys overnemen (-1 en +1 indien type = int)

Naamgeving:

* Goede naamgeving is heel belangrijk:
  + De naamgeving is een omschrijving van wat je gaat testen. De naam vormt een zin.
  + De naamgeving bestaat uit:
    - Naam\_klasse
    - Test\_omschrijving
    - Welke\_uitkomst\_je\_gaat\_krijgen (Exception of naam\_klasse)

Wat testen we?

**Getters, setters en private** methodes kunnen we **NIET** testen. Getters en Setters worden getest via de constructor.

Type van de methodes:

* Elke methode van de JUnit is een **VOID**-methode.
* Ook plaats je in de testklasse altijd boven de testklasse **@Test**.

Een testMethode wordt in UML omschreven als een ~

Triple A(AA) pattern:

* Arrange:
  + Je brengt alles klaar om de test te kunnen doen
    - met (declatratie en initalisatie variabele)
* Act
  + Aanroep doen naar de testklasse (1 lijntje code)
    - met (naam\_klasse naam\_object = new naam\_klasse (variabele)
* Assert
  + Check of de code heeft gedaan wat hij zou moeten doen
    - (met assertEquals(aantal, object.getNaam\_Variabele).

Alles assertions:

* **AssertEquals** (expected, actual)
  + Is de waarde gelijk aan …
* **AssertNull**(actual) & **AssertNotNull**(actual)
  + Is de waarde leeg of juist niet leeg?
* **AssertTrue**(condition) & **AssertFalse** (condition)
  + Is de boolean true of niet)
* **AssertThrows**(IllegalArgumentException.class, () -> new naam\_klasse)
  + Kijk of er Exception geworpen wordt

@ParamaterizedTest:

Dit wordt gebruikt wanneer je een paramter doorgeeft in de methodes.

Hiervoor moet je een ValueSource gebruiken om alle waarden in te stoppen.

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Geparameterizeerde testen:

NullSource

Indien je null wil aanleveren

… verder aanvullen

Double.MIN\_VALUE:

Werk je met doubles en floats gebruik dan Double.MIN\_VALUE() als grenswaarde.

Tip:

Als je verschillende waarden moet combineren (bv. lengte en breedte) kies dan bij negatieve testen enkel 1 van de waarden uit een ongeldige EP)

* Lengte: -3 Breedte: 30.2 …
* Lengte: 0 Breedte: 30.2 …

Regel van logica:

Je schrijft nooit **logica** binnenin een testmethode. (10 – 10 \* 50 / 100 🡪 10)

Korte methode schrijven van een testmethode:

void Schaal\_GeldigPercentage\_Schaalrechthoek()

Rechthoek r = new Rechthoek(10,20)

@ParamaterizedTest

@ValueSource(type (bv. int**s**) = {-5, 0, 11, 20} )

Dit geeft één methode weer waarbij deze lijst van waarden wordt gebruikt.

Cyclus van UnitTesten (enkel @Test en @BeforeEach kennen, maar cyclus zelf weten):

* Afbeelding met tekst, Lettertype, schermopname, Elektrisch blauw

  Automatisch gegenereerde beschrijving**@BeforeEach (belangrijkste methode - Setup)**
  + Een methode die wordt uitgevoerd **vóór** de uitvoering van **elke** **test** methode

Bij objecten met een String mag de waarde niet NULL zijn, dus test dit met de UnitTest.

Integer.MIN\_VALUE en Integer.MAX\_VALUE bij ints:

Dit zijn de kleinst mogelijke en grootst mogelijk waardes.

Examenvraag:

Je moet een **tabel** kunnen maken zoals **Stefaan De Ridder** altijd maakte in de les.

Postcode Oostende = 8400

Naam methode:

**Constructor**: Maak …

**Methode**: naam\_methode/omschrijving/

**Indien:postief** 🡪 Wat er gaat gebeuren

**Indien:negatief** 🡪 WerptException

## Testen maken en implementeren

Goede naamgeving:

* Maak je de testmethode van een **constructor**? Dan ziet je naamgeving er als volgt uit:
  + Deel 1: **naam** van de **constructor**
  + Deel 2: **voorwaarde** van de testmethode
  + Deel 3: het verwachte **resultaat** van deze testmethode
* Maak je de testmethode van een **methode**? Dan ziet je naamgeving er als volgt uit:
  + Deel 1: naam van de **methode**
  + Deel 2: **voorwaarde** van de testmethode
  + Deel 3: het verwachte **resultaat** van deze testmethode

Paramaterized test:

Werk je meerdere testmethoden die exact dezelfde code hebben, waarvan het getalletje verschilt, maak dan een **@ParameterizedTest.**

* Bij methodes die een **positief** resultaat hebben zal dit af en toe gaan om toe te voegen.
* Bij methodes die een **negatief** resultaat hebben zal dit bijna altijd gaan om toe te voegen.

Tips naamgeving van Deel 2: boundry’s zijn: 0 en 10

* Gebruik je 2 random **goedgekeurde getallen** tussenin de boundry’s?
  + naam\_eigenschap **Positief**
* Gebruik je de **meest rechtse boundry value**?
  + naam\_eigenschap NetGrootGenoeg
* Gebruik je de **meest linkse boundry value**?
  + naam\_eigenschap NetKleinGenoeg
* Gebruik je **één meer** dan de meest rechts boundry
  + naam\_eigenschap NetTeGroot
* Gebruik je **één minder** dan de meest linkse boundry
  + naam\_eigenschap NetTeKlein
* Gebruik je **een random getal rechts van de boundry**
  + naam\_eigenschap TeGroot
* Gebruik je **een random getal links van de boundry**
  + naam\_eigenschap TeKlein
* Gebruik je een @ParamaterizedTest voor **positieve waarden**
  + naam\_eigenschap Geldige/GoedeGrootte…
* Gebruik je een @ParamterizedTest voor **negatieve waarden**
  + naam\_eigenschap Ongeldig/teKlein

Maken van het uit te testen object in de testmethode:

NaamKlasse naamObject = new NaamKlasse (getal/waarde)

Voorbeeld: Waterfles wf = new Waterfles(1000, 500);

Invulling van de testmethode zelf:

Eenmaal de naam van de testmethode is gemaakt, moeten we hem ook echt invullen.

* Zijn de waarden in de methode **positief** dan gebruik je
  + **assertEquals(getal, naamObject.getNaamObject)**
  + **Voorbeeld: assertEquals(10, wf.getInhoud())**
* Zijn de waarden in de methode **negatief** dan gebruik je
  + **assertThrows(IllegalArgumentException.class, (), -> new naamClass(getal)**
  + **assertThrows(IllegalArgumentException.class (), -> new Waterfles(500)**

Boundry’s bij doubles (kommagetallen):

De kleinst mogelijke waarde na een boundry bij een double is **Double.MIN\_VALUE**