

Inhoudsopgave: Datum:

[1 Hoofdstuk 1: Overerving Deel 2 2](#_Toc161749687)

[1.1 Herhaling + Verduidelijking 2](#_Toc161749688)

[1.2 Klasse Object 3](#_Toc161749689)

[2 Hoofdstuk 2: Polymorfisme & Interfaces 4](#_Toc161749690)

[2.1 Skeletten van code: 6](#_Toc161749691)

[3 Hoofdstuk 4: Exceptions 8](#_Toc161749692)

[4 Hoofdstuk 5: JavaFX 9](#_Toc161749693)

[4.1 GUI codes 10](#_Toc161749694)

[Examenvragen: 11](#_Toc161749695)

# Hoofdstuk 1: Overerving Deel 2

## Herhaling + Verduidelijking

Schema van alle mogelijk datatypes:

Afbeelding met tekst, schermopname, lijn, Lettertype

Automatisch gegenereerde beschrijving

Bij het maken van objecten/referenties van variabelen komen, zoeken deze waarden plaats op in het **RAM** en wordt het daar opgeslagen.

Communiceren met objecten via de *de-reference* operator:

Om te spreken met objecten gebruiken we altijd een **.** om dit te doen.

Afbeelding met tekst, schermopname, diagram, lijn

Automatisch gegenereerde beschrijvingIS-EEN Relatie (BOTTOM TO TOP):

Een overerving van een superklasse is altijd een **IS-EEN relatie**.

* Rechtoek **IS EEN** figuur.
* Driehoek **IS EEN** figuur.

KAN EEN Relatie (TOP TO BOTTOM):

* Object **KAN EEN** Figuur zijn.
* Rechthoek **KAN EEN** Vierhoek zijn.

Alle soorten verbindingslijnen in Visual Paradigm:

Afbeelding met tekst, schermopname, lijn, diagram

Automatisch gegenereerde beschrijving

Afbeelding met schermopname, tekst, Lettertype

Automatisch gegenereerde beschrijving

SUPER-klasse:

* De superklasse wordt **altijd** aangeroepen, of je het er nu bijschrijft of niet, hij staat er altijd.
* Is het geen superklasse van overerving, wordt het wel altijd aangeroepen vanuit de klasse **Object**.

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

Visibility protected:

Dit attribuut kan enkel gebruikt worden in **dezelfde** **package** of Een Superklasse ervan.

Default visibilty Java:

Packagewidevisibility

Instanceof:

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

## Klasse Object

Object methoden:

* Equals: definieert de notie van **gelijkheid** gebaseerd op de toestand van een object, los van de **referentie**.
* Tostring: geeft een **string representatie** terug van het object.
* Getclass: geeft een **referentie** terug naar het **class object**, een representatie van de klasse van dat object.

Abstracte klassen:

* Je geeft aan dat het niet de bedoeling is om **instanties** te maken van de **superklasse**, enkel van de **subklassen**, maar je kan wel zeggen **Rekening rekening = new Zichtrekening** (geen rekeningen, maar spaar- of zichtrekening).
* Abstracte klassen hebben ook **geen** **body** in de superklasse.
* 1 abstracte methode = volledige klasse abstract
* Abstracte klassen/methoden in **UML** = **schuingedrukt**.
* **Dit is de reden waarom we de abstracte methode in superklasse Rekening hebt gezet.**

Afbeelding met tekst, Lettertype, Multimediasoftware, schermopname

Automatisch gegenereerde beschrijving

Object functie: equals:

Dit laat je zelf geneneren. Vergelijkt dat 2 of meerdere objecten met elkaar gelijk zijn.

# Hoofdstuk 2: Polymorfisme & Interfaces

Final methodes:

Kan je niet **meer overschrijven**, kan de body niet meer veranderen.

Final attribuut:

Kan de **waarde** niet meer veranderen.

Herhaling abstracte klassen:

* Je kan **geen objecten** van maken van de superklasse (Motor, elektrischeMotor, bezineMotor) Hier kan Motor niet gemaakt worden omdat hij abstract is.
* Methodes die je hier inschrijft kan je overal **wel** gebruiken.

Interface:

* Je hebt **geen eigenschappen/attributen**
* Je hebt **enkel abstracte methodes** (geen body/invulling)
* Kenmerk van een interface is een kleine **“I”** boven de naamklasse.
* In methodes zitten het woord “**public abstract**” **vast ingebouwd** onzichtbaar, maar je mag het wel nog is noteren als je wil.
* Bij **overervering** gebruik je het woord “**extends**”, bij **interface** gebruik je het woord “**implements**”
* Je bent **verplicht** alle abstracte methodes uit de interface te gebruiken.
* Een klasse kan **meerdere interfaces implementeren**, overerving slechts één.
* Je kan geen interface **instantiëren**
* **Je maakt een instantie van een klasse van de interface dat je implementeert.**

Verschil overerving en interface in UML:

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

Comparable interfaces (Interface Comparable <T>):

Als je niet kan vergelijken, kan je niet sorteren.

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijvingFunctionele interface:

* Slechts één abstracte methode: compareTo
* Negatief getal f1 < f2
* Positief getal f2 f1
* nul f2.equals(f1)
  + Vast skelet



Inteface als type:

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijvingEen interface kan ook gebruikt worden als type.

Natuurlijke ordening:

* I comparable
* In klasse zelf
* Compareto
* Consistentie met equals/hashcode
* Één comparable mogelijk **!EXTREEM BELANGRIJK!**

Totale ordening:

* I Comparator
* Compare
* Verschillende comparebles mogelijk
* Buiten klasse zelf

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijvingFunctionele interface:

Interface met exact 1 asbolute methode

Anonieme innerklasse:

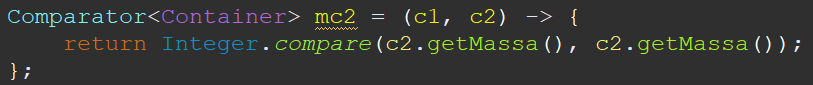
* Het heeft **geen naam**
* innerklasse = in een andere klasse
* Je maakt een instantie van een klasse die geen naam heeft/die comparator implementeert.

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Lambaexpressie:

* ->
* Dit kan je enkel gebruik met functionele interfaces
* Vereenvoudiging van de functie hierboven



CompareTo:

* m1.compareTo(m2);
  + Is m1 kleiner dan m2? = -1
  + Is m1 groter dan m2? = 1
  + Is m1 gelijk aan m2? = 0

Ingebouwde functie comparing:



## Skeletten van code:

Lambada-expressies:

Eenvoudigere versie van een Comparator (meerdere versies):

Comparator<Movie> rcOld = Comparator.*comparing*(m -> m.getRating());

De meest eenvoudige versie van een Comparator (oplopend en aflopend)

Comparator<Movie> rc2 = Comparator.*comparing*(Movie::getRating);

Comparator<Movie> rc2 = Comparator.*comparing*(Movie::getRating).reversed();

Hashcode en equals:

Je genereert beide op basis van welke precieze eigenschap je wil vergelijken

Dan implementeer je implements Comparable<naamParameter> naar jouw keuze

Daarna implementeer je de methode die daarna nog ontbreekt, hier heb je meerder skeletten van:

Vergelijken met één iets:

if (this.jaar > f.jaar)

return -1;

if (this.jaar < f.jaar)

return 1;

return 0;

Vergelijken met één String:

public int compareTo(Film f) {

return this.naam.compareTo(f.naam);

}

Eerst sorteren op datum film, indien ze gelijk zijn, sorteren op datum:

public int compareTo(Film f) {

int naamVergelijk = this.naam.compareTo(f.naam); // Omgekeerd sorten = \*-1

if (naamVergelijk != 0)

return naamVergelijk;

return f.jaar - this.jaar;

}

# Hoofdstuk 4: Exceptions

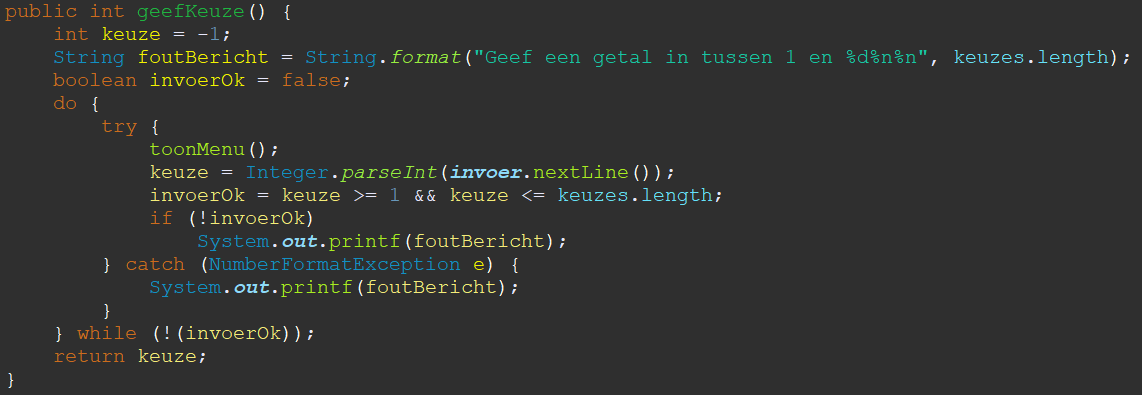
Checked exception:

* Exception
  + Deze moet je opvangen in je code of je throwt hem

Unchecked exception:

* IllegalArgumentException

Nieuwe manier van Scanner:



Herhaling eigen Exception maken:

* Nieuwe package “exceptions”
* Laten erven van IllegalArgumentException + onderste 2 keuzes aanvinken
* Bovenste of onderste IllegalArgument invullen met tekst
* Nu is je exception af, implementeer validatie in domein/cui en throw je zelfgemaakte exception

# Hoofdstuk 5: JavaFX

Afbeelding met tekst, schermopname, software, Multimediasoftware

Automatisch gegenereerde beschrijvingPackage main:

* StartUp laten we vanaf nu erven van Application
* Dit genereert een venster voor jou

Maken van een nieuw project:

Vanaf nu als we een project maken selecteren we **“module.info.java”** die de klasse van onze naam bevat maar dan met een kleine letter.

Soorten vensters:

* Pane
* GridPane
* VBox
* HBox
* BorderPane
* FlowPane

CSS:

CSS werkt hier net iets anders in vergelijking met html, hier begint alle code met **-fx-**

## GUI codes

Opmerkingen:

* Volgorde maakt enorm veel uit
* **Alles** is **aanpasbaar** en alles heeft veel **verschillende types**
* Het is de bedoeling dat jij zelfstandig de GUI ontdekt.

Button:

Button btnOk = new Button("Ok");

root.getchildren(): Toevoegen van dingen aan de rootpanel

root.getChildren().addAll(lblWelkom, btnOk);

Label:

Label label = new Label("Welkom");

Padding geven:

root.setPadding(new Insets(20));

Spaties tussen objecten:

root.setSpacing(20);

Event handler:

btnOk.setOnAction((evt) -> {

System.***out***.println("Bedankt voor het klikken");

});

Event handler v2 met alert:

btnOk.setOnAction((evt) -> {

Alert artThanks = new Alert(*AlertType*.***INFORMATION***, "Bedankt voor het klikken");

artThanks.showAndWait(); });

## Opmaak.css

.label geeft opmaak op alle labels

# Examenvragen:

Maak je eigen Exception:

* Maak een nieuwe class in package exceptions
* Extend de klasse met “java.lang.exception”
* Genereer de superklasse het meest onderste vinkje
* Enkel de eerste twee constructors zijn goed genoeg om aan te vinken

Ordenen:

* Genereer