Cursus Robbe Magerman Object Oriënted Software Development 2

Robbe Magerman

03/06/2024

Inhoudstafel

1 Handige dingen

1.1 Objecten omzetten naar DTO's

```
    private Collection<ContainerDTO> zetContainersOmNaatContainerDTOs(
       Collection<Container> containers) {
        Collection<ContainerDTO> containerDTOs = new ArrayList<>();
        for (Container c : containers) {
            containerDTOs.add(new ContainerDTO(c));
        }
        return containerDTOs;
    }
```

1.2 Constructor van een DTO

```
package dto;

import domein.Hoofdgerecht;

public record HoofdGerechtDTO(String naam, double prijs, String type) {

    public HoofdGerechtDTO(Hoofdgerecht gerecht) {
        this(gerecht.getNaam(), gerecht.getPrijs(), gerecht.getType());
    }
}
```

1.3 Mapperklasse

```
package persistentie;

import java.util.ArrayList;
import java.util.List;

import domein.Hoofdgerecht;

public class HoofdgerechtMapper {

    public List<Hoofdgerecht> geefHoofdgerechten() {
        List<Hoofdgerecht> lijstMetHoofdGerechten = new ArrayList<>();
        lijstMetHoofdGerechten.add(new Hoofdgerecht("Bolognaise", 7.51, "pasta"
           ));
        lijstMetHoofdGerechten.add(new Hoofdgerecht("Spinazie met worst", 6.50,
            "aardappelen"));
        lijstMetHoofdGerechten.add(new Hoofdgerecht("Hamburger du chef", 9.99,
           "burger"));
        lijstMetHoofdGerechten.add(new Hoofdgerecht("Spaghetti Carbonara",
           11.50, "pasta"));
        lijstMetHoofdGerechten.add(new Hoofdgerecht("Frietjes met stoofvlees",
           11.50, "aardappelen"));

        return lijstMetHoofdGerechten;
    }
}
```

1.4 DomeinController-klasse

```
package domein;

import java.util.List;

import persistentie.HoofdgerechtMapper;
import persistentie.VoorgerechtMapper;

public class DomeinController {

    private HoofdgerechtMapper hoofdGMapper;

    public DomeinController() {
        this.hoofdGMapper = new HoofdgerechtMapper();
    }

    public List<Hoofdgerecht> geefAlleHoofdgerechten() {
        return hoofdGMapper.geefHoofdgerechten();
    }
}
```

2 Hoofdstuk 1

2.1 Abstracte classes

2.1.1 Wat?

- `public abstract class Rekening` Is de basis van een abstracte klasse
- Eenmaal dit geïmplementeerd **moeten** alle kinderen die van deze klasse `extends` die methodes zeker gebruiken.
- In deze *Rekening* klasse moet je een methode in zetten die andere klassen moeten gebruiken: `public abstract int geldOverdracht`
- Een abstract klasse mag een volledige invulling hebben, het is namelijk geen interface

3 Hoofdstuk 2

3.1 Interfaces

3.1.1 Wat?

`public interface Beheerstkost`

- Eenmaal dit geïmplementeeerd **moeten* alle kinderen die van deze klasse `implements` de gegegeven methodes implementeren via een `@Override`

4 Hoofdstuk 3 Lamda Expressies

4.1 Comparators - CompareTo

4.1.1 Comparator Algemeen

- `class Movie implements Comparable<Movie>`
- `hashcode` en `equals` genereren

4.1.2 Comparator via een String

```
@Override
public int compareTo(Movie m) {
    int compareName = name.compareTo(m.name);
    return compareName != 0 ? compareName : year - m.year;
}
```

4.1.3 Comparator via een int

```
@Override
    public int compareTo(Container c) {
        return Integer.compare(this.serialNumber, c.serialNumber);
    }d
```

4.1.4 Comparator via eerst een int daarna een String

```
@Override
    public int compareTo(Auto a) {
        int result = Integer.compare(aantalOnderhoudsbeurten, a.
           aantalOnderhoudsbeurten);
        if (result != 0)
            return result;
        return nummerplaat.compareTo(a.nummerplaat);
    }
```

4.1.5 Comparator met een datum

```
@Override
    public int compareTo(Film f) {
        if (this.jaar > f.jaar)
            return -1;
        if (this.jaar < f.jaar)
            return 1;
        return 0;
    }
```

4.2 Comparators - Compare

4.2.1 Comparators in een andere klasse - Algemeen

- Net zoals we een gewone klasse maken in een Domein maken we dat ook voor een comparator
- We **implementeren** de klasse waarin we gaan comparen:
  - `public class MassacComparator implements Comparator<Container>`

- Hier is een volledig voorbeeld van zo een klasse:

```
package domein;
import java.util.Comparator;
public class RatingComparator implements Comparator<Movie> {
    @Override
    public int compare(Movie m1, Movie m2) {
        return Double.compare(m1.getRating(), m2.getRating()) * -1;
    }
}
```

4.2.2 Comparators in een andere klasse via een int

```
@Override
    public int compare(Container c1, Container c2) {
        return Integer.compare(c1.getMassa(), c2.getMassa());
    }
```

4.2.3 Comparators in een andere klasse via een String

```
@Override
    public int compare(Container c1, Container c2) {
        return c1.getEigenaar().compareTo(c2.getEigenaar());
    }
```

4.2.4 Comparators in een andere klasse via een volume, daarna een String

```
@Override
    public int compare(Container c1, Container c2) {
        int volumeCompare = Integer.compare(c1.getVolume(), c2.getVolume());
        if (volumeCompare != 0)
            return volumeCompare;
        return c1.getEigenaar().compareTo(c2.getEigenaar());
    }
```

4.3 Lambdas Expressies - methodreferencing

- Je haalt via een Lambdas expressie rechtstreeks de methode op van een klasse waarmee je wil comparen

4.3.1 Comparen met een merk daarna een model (String en String)

```
Collections.sort(autos, Comparator.comparing(Auto::getMerk).thenComparing(Auto
    ::getModel));
```

4.3.2 Comparen via een merk (String)

```
Collections.sort(autos, Comparator.comparing(Auto::getMerk));
```

4.4 Comparator - Anonieme innerklasse

4.4.1 Een comparator van het domein zelf

- Je roept een functie op

```
Comparator<Movie> yearComp1 = new Comparator<Movie>() {
            @Override
            public int compare(Movie m1, Movie m2) {
                return Integer.compare(m1.getYear(), m2.getYear());
            }
        };
```

4.4.2 Een comparator aan de hand van een domeinklasse

```
Collections.sort(allMovies, new YearComparator());
        showMovies("Movies with total ordering based on year", allMovies);
```

4.4.3 Extra voorbeeld van een anonieme innerklasse

```
    // Anonieme klasse
    public Collection<Container> geefAlleContainersGesorteerdOpMassa() {
        Collections.sort(containers, new Comparator<Container>() {
            @Override
            public int compare(Container o1, Container o2) {
                return Integer.compare(o1.getMassa(), o2.getMassa());
            }
        });
    }
```

4.4.4 Collections.sort

```
  List<String> namenLijst = Arrays.asList(new String[] { "Ena", "Cho", "Aiko",
     "Rai" });
  Collections.sort(namenLijst, (n1, n2) -> n1.compareTo(n2));
```

5 Hoofdstuk 4: Exceptions en robuustheid

5.1 Exceptions in aparte classes

- Maak een nieuwe class aan met `naamKlasseException`
- Vink `constructors van superclass aan`
  - Hiervan zul je meestal de 1e (en de 2e) constructor gebruiken
- De tekst die je ingeeft is de default tekst, je kan die aanpassen door in de klasse waar je hebt oproept zelf een tekst in te schrijven
- Je moet hem ook laten erven van een bepaalde soort Exception, vb. :

```
public class ContainerException extends Exception {

    public ContainerException() {
        super("Fout");
    }

    public ContainerException(String message) {
        super("fout bij de container" + message);
    }
}
```

5.1.1 Throw Declaraties

- Je kan ook een methode gebruik laten maken van een `throws naamClassException`
- Deze throws zal zich dan voorterven naar elke klasse die van elkaar zullen gebruik maken
  - Het start bv. in klasse 1, gaat door naar klasse 2, 3, 4 ... en zo naar de startUp
- Hiervoor gebruik je in je exceptionbestand niet de parameterloze constructor, maar wel degenen met `(String message)`

5.2 Volledig Robuust menu

5.2.1 Header van een Menu

```
    public int kiesUitMenu() {
        String foutmelding = String.format("%n === Gelieve een getal tussen 1
            en 3 in te voeren...%n ===");
        boolean geldigeInvoer = false;
        int keuze = -1;
        do {
            try {
                System.out.printf("1. Toon een overzicht van alle producten %n2
                    . Voeg een plant Toe%n3. Afsluiten%n");
                System.out.printf("Maak je keuze: ");
                keuze = Integer.parseInt(invoer.nextLine());
                geldigeInvoer = keuze >= 1 && keuze <= 3;
                if (!geldigeInvoer) {
                    System.out.println(foutmelding);
                }
            } catch (NumberFormatException e) {
                System.out.printf(foutmelding);
            } catch (IllegalArgumentException e) {
                System.out.println(" === Er ging nog iets harder fout ===");
            }
        } while (!geldigeInvoer);
        return keuze;
    }
```

5.2.2 Robuust Menu via een double

```
public double geefPrijsPlant() {
    String foutmelding = "\n === Geef een geldig nummer in ===";
    boolean geldigeInvoer = false;
    double prijs = 0;
    do {
        try {
            System.out.println("Geef de prijs van de plant op: ");
            prijs = Double.parseDouble(invoer.next());
            geldigeInvoer = prijs > 0;
            if (!geldigeInvoer)
                System.out.printf("%s", foutmelding);
        } catch (NumberFormatException e) {
            System.out.println("\n === Dit is geen getal ===");
        } catch (IllegalArgumentException e) {
            System.out.println("\n === Iets ging fout in verband met al de
               rest ===");
        }
    } while (!geldigeInvoer);
    return prijs;
}
```

5.2.3 Robuust Menu via een String

```
private String geefNaamPlant() {
    String naam = "";
    String foutmelding = ("\n === Geef een geldige gebruikersnaam in ===");
    boolean geldigeInvoer = false;
    do {
        try {
            System.out.print("Geef de naam van de plant in: ");
            naam = invoer.nextLine();
            // groter dan 2, niet leeg
            geldigeInvoer = naam.length() > 2 && !naam.isBlank() && !naam.
               isEmpty();
            if (!geldigeInvoer)
                System.out.println(foutmelding);
        } catch (IllegalArgumentException e) {
            System.out.println(foutmelding);
        }
    } while (!geldigeInvoer);
    return naam;
}
```

5.2.4 Robuust menu door ja of nee te vragen

- Het belangrijkste is hier de `equalsIgnoreCase`

```
private boolean vraagJaNeeBevestiging(String vraag) {
    String naam;
    boolean invoerOK;
    do {
        System.out.printf("%s (ja/nee): ", vraag);
        naam = invoer.nextLine();
        invoerOK = naam.equalsIgnoreCase("ja") || naam.equalsIgnoreCase("
           nee");
```

```
            if (!invoerOK)
                System.out.println("Gelieve met 'ja' of 'nee' te antwoorden..."
                    );
        } while (!invoerOK);
        return naam.equalsIgnoreCase("ja");
    }
```

6 Hoofdstuk 5: JavaFX

6.1 Hoe krijg je een startscherm:

- Maak een nieuw project aan
- Maak de `StartUp` klasse aan in `package main` en laat het *erven* van Application.

6.2 JavaFX codes

6.2.1 Button

```
buttonAfsluiten = new Button("afsluiten");
```

6.2.2 Css toevoegen

```
buttonAfsluiten.setStyle("-fx-background-color: red;");
```

6.2.3 Font toevoegen

```
// font toevoegen als bestand
Button buttonAfsluiten.setFont(Font.font("Berkshire Swash", 15));
```

6.2.4 Action op een button + afsluiten + alert

```
buttonAfsluiten.setOnAction((evt) -> {
            Alert alertAfsluiten = new Alert(AlertType.INFORMATION, "Bedankt
               voor het spelen, tot de volgende keer!");
            alertAfsluiten.showAndWait();
            Platform.exit();
        });
```

6.2.5 Image

```
ImageView imageViewFavicon = new ImageView();
        Image imageFavicon = new Image(getClass().getResourceAsStream("/gui/img
           /mascotte_rechtgedraaid.png"));
        imageViewFavicon.setImage(imageFavicon);
        this.getChildren().addAll(imageViewFavicon);
```

6.2.6 Switchen van scherm

```
getScene().setRoot(registratieScherm);
```

6.2.7 Label

```
Label label = new Label("Hier zet ik tekst op");
```

6.2.8 Hyperlink

```
Hyperlink linkForgot = new Hyperlink("Forgot password");
```

6.2.9 Padding

```
Insets padding = new Insets(10);
label.setMargin(padding);
```

6.2.10 Spacing

```
setSpacing(10);
```

6.3 JavaFX Lay-outcontainers

Box	Betekenis
`VBox`	Plaats de inhoud op een *verticale* as op elkaar
`HBox`	Plaats de inhoud op een *horizontale* as op elkaar
`GridPane`	Geeft de inhoud elk een *specifieke plaats*
`BorderPane`	Plaats de inhoud aan de *randen* van het scherm of in het midden
`Stackpane`	Plaatst de inhoud *boven elkaar* plaatst en zichzelf aanpast aan de grootte van zijn kinderen. Zo maak je bv een achtergrond op een scherm
`AnchorPane`	Plaatst de inhoud ten opzichte van de *randen* van de container of ten opzichte van elkaar door ankers te gebruiken.
`FlowPanes`	Plaatst de inhoud in een *flow van links naar rechts* en van boven naar onder. Het past automatisch de grootte van zijn kinderen aan en kan over meerdere rijen en kolommen worden verdeeld.
`TilePane`	Plaatst de inhoud in een *tegelpatroon*. Het past automatisch de grootte van zijn kinderen aan en kan worden geconfigureerd om zijn kinderen te ordenen in een vaste rij of kolom, of om ze vrij te laten stromen.

7 Hoofdstuk 6: Collecties

7.1 Theorie

- Een List is een `Collection` het returntype vanaf nu dat de DomeinController dus zal ontvangen is een `Collection`, hier zullen dus enkel methodes worden toegepaste van de `interface Collection`

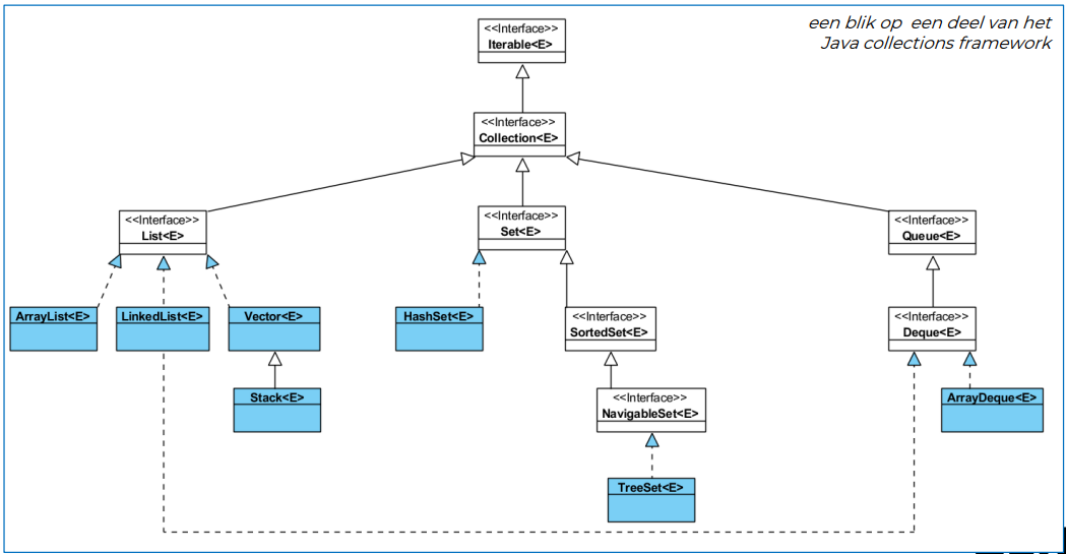


Figure 1: Structuur van de functionele interface: Collection

7.2 Interface Iterator

- Deze interface heeft 3 methodes:
  - hasNext()
    - Retourneert true als de iteratie nog elementen bevat
  - next()
  - remove()
    - Verwijdert het laatste element

7.3 Interface List

- Geordende collectie (sequence)
- Mag dubbels bevatten
- Elk element heeft een positie
  - zZero based index
  - Elementen kunnen worden opgehaald, toegevoegd en verwijderd op basis van deze index
- Voorziet in een ListIterator

Methodes:

- E get(int index)
- int indexOf(Object o)
- int lastIndexOf(Object o)
- E remove(int index)
- E set(int index, E element)
- List subList(int fromIndex, int toIndex)

7.4 Interface ListIterator

- add(E e)
- hasNext()/hasPrevious()

- next()/previous()
- nextIndex()/previousIndex()
- remove()
- set(E e)

7.5 List interface

7.5.1 ArrayList class

- Mogelijkheid om grootte (capacity) vanonderliggende array in te stellen en temanipuleren

7.5.2 LinkedList class

- Elk element heeft een verwijzing naar het volgende en het vorige element

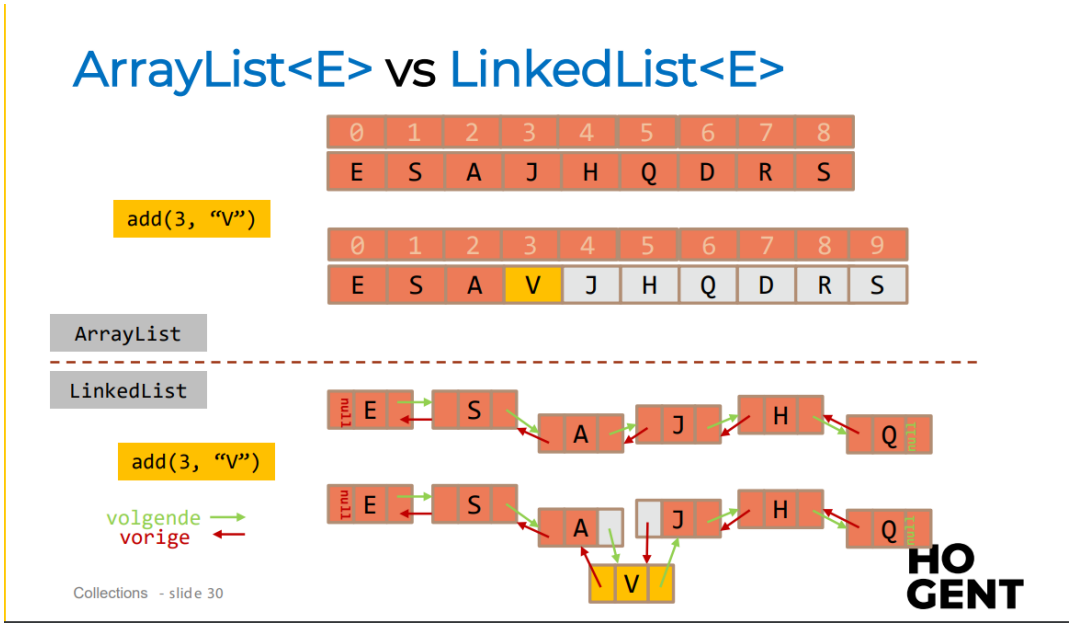


Figure 2: Verschil ArrayList en LinkedList

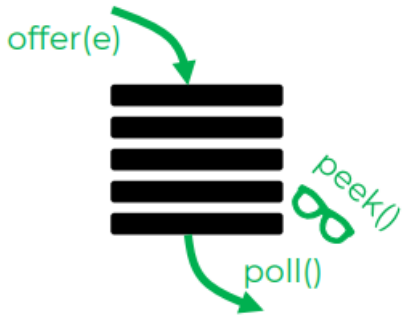
-

ArrayList	LinkedList
elementen in resizeable array	elementen in doubly linked list
constante toegangstijd voor elk element ⇒ random access	sequentiële toegang ⇒ starten vanaf eerste element
invoegen of verwijderen van element ⇒ veel verschuivingen	efficiënt toevoegen en/of verwijderen van element
bij voorkeur te gebruiken bij veel **opzoekingen**	bij voorkeur te gebruiken bij **veel invoegen/verwijderen** van elementen

Figure 3: Verschil ArrayList en LinkedList in woorden

-

7.5.3 Stack



- Stacks kan je performanter implementeren met ArrayDeque

7.5.4 Queue interface

- FIFO – First In First Out structuur
- Er bestaan twee versies van methodes om elementen toe te voegen, te bekijken, te verwijderen
  - Je maakt beter gebruik van offer/poll & peek
    - Indien de queue leeg is retourneert poll/peek de waarde null
    - Offer retourneert true als het element werd toegevoegd

	Throws exception	*Returns special value*
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

Figure 4: Alle methodes van een queue

-

7.5.5 Deque

Queue<E> Method	Equivalent Deque<E> Method	*exception or special value*
add(e)	addLast(e)	*exception*
offer(e)	offerLast(e)	*true/false*
remove()	removeFirst()	*exception*
poll()	pollFirst()	*null*
element()	getFirst()	*exception*
peek()	peekFirst()	*null*

Figure 5: alt text

7.5.6 Set

- wiskundige notie van verzameling
- mogelijks orde op de elementen
- geen toegang via index
- kan geen dubbels bevatten
- een set bevat nooit twee elementen e1 en e2 waarvoor e1.equals(e2) true is
- override equals & hashcode voor de klasse E

7.5.7 HashSet

- concrete implementatie van Set die een hashtabel als onderliggende structuur heeft
- de hashcode is bepalend voor de plaats waar een element in een hashtabel terecht komt
- elementen zijn niet geordend
- itereren is onvoorspelbaar

7.5.8 SortedSet

- een subinterface van Set die elementen geordend bijhoudt
- natuurlijke orde: compareTo
- totale orde: Comparator
- typische methodes
- E first()
- E last()
- SortedSet subset(E fromElement, E toElement)
- SortedSet headset(E toElement)
- SortedSet tailset(E fromElement)

7.5.9 NavigableSet

- een SortedSet met extra notie van nabijheid van elementen
- typische methodes
- E floor(E e)
- E higher(E e)
- E lower(E e)
- E ceiling(E e)

7.6 Arrays

7.6.1 asList() - From array to collection

- Deze List is fixed size en heeft de array als onderliggende structuur, m.a.w. veranderingen in de lijst worden ook doorgevoerd in de oorspronkelijke array, en omgekeerd...

```
String[] wordsInArray = { "Arrays", "are", "fun" };
List<String> wordsAsList = Arrays.asList(wordsInArray);
wordsAsList.set(0, "Lists");
System.out.println(wordsInArray[0]); // prints "lists"
wordsAsList.add("!");
```

7.6.2 toArray() - From collection to array

```
Set<String> wordsInSet = new HashSet<>();
wordsInSet.add("Java");
wordsInSet.add("Python");
String[] wordsInArray = wordsInSet.toArray(new String[5]);
wordsInSet.remove("Java");
```

```
Set<String> wordsInSet = new HashSet<>();
wordsInSet.add("Java");
wordsInSet.add("Python");
String[] wordsInArray = wordsInSet.toArray(new String[0]);
```

7.7 Class Collections

- klasse met verschillende static methods om collections te manipuleren
  - sort
  - search
  - copy
  - reverse, shuffle, replace, ..
- binarySearch(...)
  - zoeken in een collection, of in een deel ervan, naar een specifiek element
    - copyOf(...)/copyOfRange(...)
  - een copy nemen van een collection, of een deel ervan
    - fill(...)
      - het vullen van een collection, of een deel ervan, met een specifiek element
    - sort(...)
      - sorteren van een collection, of een deel ervan, gebruik makend van compareTo (natuurlijke ordening) of een comparator (totale ordening)

7.7.1 ArrayAsList in een domeinlaag

- Dit werkt niet als een gewone (dynamische) array waarbij je dingen kan toevoegen

```
    private final List<Stripfiguur> stripfiguren;

    stripfiguren = new ArrayList<>(Arrays.asList(mapper.geefStripfiguren()));
```

7.7.2 ArrayAsList van een normale simpele lijst

- Hier is SUITS de oorspronkelijke lijst

```
        List<String> list = Arrays.asList(SUITS);
```

7.8 Structuren

7.8.1 Collections.sort()

- eerst maken we een hashcode en equals in de `domeinklasse`
- Daarna maken we een compare en typen we dit in de `repository`

```
    @Override
    public int compareTo(Stripfiguur f) {
        int naamComparison = naam.compareTo(f.naam);
        return naamComparison != 0 ? naamComparison : Double.compare(grootte, f
            .grootte);
    }
```

- Waarbij we daarna dit implementeren in de Repository

```
    public Collection<Stripfiguur> geefStripfigurenGesorteerdOpNaam() {
        Collections.sort(stripfiguren);
        return stripfiguren;
    }
```

- We kunnen deze klasse ook uitbreiden met een lambda expressie:

```
    public Collection<Stripfiguur> geefStripfigurenGesorteerdOpNaam() {
        Collections.sort(stripfiguren, Comparator.comparing(Stripfiguur::
            getNaam).reversed());
        return stripfiguren;
    }
```

- alle stripfiguren zonder dubbels

```
    public Collection<Stripfiguur> geefStripfigurenZonderDubbels() {
        Collection<Stripfiguur> stripfigurenZonderDubbels = new HashSet<>(
            stripfiguren);
        return stripfigurenZonderDubbels;
    }
```

- Drie willekeurige stripfiguren

```
    public Collection<Stripfiguur> geefDrieWillekeurigeStripfiguren() {
        Collections.shuffle(stripfiguren);
        return stripfiguren.subList(0, 3);
    }
```

- Iets toevoegen aan een lijst

```
    public void voegStripfiguurToe(String naam, double grootte) {
        stripfiguren.add(new Stripfiguur(naam, grootte));
    }
```

- Overstappen van array naar Collection

```
// normale methode
    stripfiguren.addAll(Arrays.asList(mapper.geefExtraStripfiguren()));

// stripfiguren toevoegen zonder dubbels3+
    HashSet<>(Arrays.asList(mapper.geefExtraStripfiguren()));
```

8 Hoofdstuk 7:

8.1 Theorie

8.1.1 Een stream via de functionele interface - met ints

```
int[] values = { 3, 4, 6, 1 };
IntStream stream1 = IntStream.of(values);
```

```
List<String> dataSource = new ArrayList<>(Arrays.asList("yellow", "green", "
    red", "orange"));
Stream<String> stream = dataSource.stream().filter(s -> s.length() > 5);
```

8.1.2 Een stream via een gemaakte Lijst

```
    stripfiguren = new ArrayList<>(Arrays.asList(mapper.geefStripfiguren()));
    return stripfiguren.stream().sorted(Comparator.comparing(Stripfiguur::
        getNaam)).toList();
```

8.1.3 imperatief (forlus in java)

Hoe en wat, explicitate iteratie

```
for (Employee e : employees){
    if (e.getSalary() > 10000)
        richEmployeesNames.add(e.getName());
}
```

8.1.4 declaratief (select in sql)

Wat, geen explicitie iteratie

```
select name
from eployees
where salary > 10000
```

8.1.5 Interface Stream

- Een Stream is een `sequentie` van objecten waarop geaggrgeerde en parallelle operaties kunnen toegepast worden
- We slaan er geen data in op
- We gebruiken het om iets te doen op de data
- Stap 1: we vormen een `Collection` / `Array` naar een Stream
- Stap 2: toevoegen van operaties die opnieuw een Stream opleveren (mapper, filteren . . .) we noemen dit ook intermediate opertations
  - Dit heeft als invoer een Stream en uitvoer ook
  - Bij een Terminal operation gaan we uit de Stream
- Stap 3: We gaan uit de Stream wereld en komen weer terug in de Java wereld

8.1.6 . . . Stream().of()

- Je moet kijken naar het type waarop je een stream toepast
  - int? –> intStream().of()
  - Objecten? –> stream().of()

8.1.7 Arrays.stream && Arrays.asList zeer belangrijk

- Overstappen van Array naar Streams
- Overstappen van Array naar Collection

8.1.8 verschil tussen map en mapTO...

```
.map(e -> getSalary())
    Stream<Double>

.mapToDouble(e -> e.getSalary())
    DoubleStream
```

8.1.9 Filter

- Data die aan een bepaalde voorwaarde doet

```
int[] values = {3, 4, 6, 1 }
IntStream stream= IntStream.of(values).filter(i -> i % 2 == 0);
```

8.1.10 map

-

```
int[] values = {3, 4, 6, 1 }
IntStream stream= IntStream.of(values).map(i -> i / 2);
```

8.1.11 distinct

- Verwijdert alle dubbele waarden, (gebruikt de equals methode in de achtergrond - niet hetzelfde als Comparator)

```
int[] values = {3, 4, 6, 1 }
IntStream stream= IntStream.of(values).distinct(.).filter(i -> i % 2 == 0);
```

8.1.12 Sorted

- Zorgt ervoor dat de uitkomst van de waarden gesorteerd is

```
int[] values = {3, 4, 6, 1 }
IntStream stream= IntStream.of(values).sorted();
```

8.1.13 OptionalInt

- Dit bevat mogelijk een waarde
- Deze klasse wordt gebruikt bij de max() functie

```
- int getAsInt() // Double, Long
- int orElse(int other)
- boolean isPresent()
```

8.1.14 findFirst()

- Dit wordt gebruikt bij streams
- Wanneer dit wordt gebruikt, wordt het 1e 'object' dat aan die voorwaarde voldoet weergegeven

8.1.15 reduce()

- Hier wordt een berekening teruggegeven
- Er is geen previousValue

```
.reduce(x, y) -> x + y
// of
.reduce (0, (x, y) -> x + y)
```

8.1.16 Collectors.toCollections

- Hier kan je alles inzetten wat je kent van Collections
- bv.

```
.collect(Collecters.toCollections(HashSet::new))
```

8.1.17 anyMatch()

- Werkt als de filtermethode, kijkt of iets aan een bepaalde voorwaarde voldoet en levert *automatisch* true of false op

8.2 Oefeningen

8.2.1 Elementen verzamelen in een Collection

```
    public Collection<Vliegmaatschappij> geefMaatschappijenMetPartners(int
        minAantal) {
         return maatschappijen.stream().filter(vm -> vm.getPartners().size() >=
            minAantal)
                 .collect(Collectors.toCollection(ArrayList::new));
    }
```

8.2.2 elementen alfabatisch gesorteerd zonder natuurlijke ordening

```
    public Collection<Vliegmaatschappij> geefAirlinesAlfabetischGesorteerd() {
         return maatschappijen.stream().sorted().collect(Collectors.toCollection
            (ArrayList::new));
    }
```

8.2.3 Elementen omzetten naar een DTO

```
    private Collection<VliegmaatschappijDTO> geefVliegmaatschappijDTOs(
        Collection<Vliegmaatschappij> maatschappijen) {
         return maatschappijen.stream().map(vm -> new VliegmaatschappijDTO(vm))
                 .collect(Collectors.toCollection(ArrayList::new));
    }
    // of
    private Collection<VliegmaatschappijDTO> geefVliegmaatschappijDTOs(
        Collection<Vliegmaatschappij> maatschappijen) {
    return maatschappijen.stream().map(vm -> VliegmaatschappijDTO(vm)::new)
             .collect(Collectors.toCollection(ArrayList::new));
    }
```

8.2.4 Alle DTO-objecten aan elkaar koppelen in de ui

```
    private String geefAlleDTOsInEenString(Collection<VliegmaatschappijDTO>
        vmDTOs) {
        return vmDTOs.stream().map(vm -> dtoToString(vm)).collect(Collectors.
            joining());
    }
```

8.2.5 Gestorteerd op aantal partners

```
    public Collection<Vliegmaatschappij>
        geefAirlinesGesorteerdVolgensAantalPartners() {
        return maatschappijen.stream().sorted(Comparator.comparing((
            Vliegmaatschappij v) -> v.getPartners().size())
                .thenComparing(Vliegmaatschappij::getNaam)).collect(Collectors.
                    toCollection(ArrayList::new));

    }
```

8.2.6 Iets teruggeven startend met ...

```
    public String geefEersteAirlineStartendMet(String startTekst) {
        return maatschappijen.stream().filter(vm -> vm.getNaam().startsWith(
            startTekst)).findFirst()
                .map(vm -> vm.getNaam()).orElse("niet bestaande");
    }
```

8.2.7 Elementen met een aantal ...

```
    public Vliegmaatschappij geefEenAirlineMetPartner(String partner) {
        return maatschappijen.stream().filter(vm -> vm.getNaam().contains(
            partner)).findAny().orElse(null);
    }
```

8.2.8 toList

- Een niet wijzigbare lijst maken

8.2.9 Hulpmiddelen:

```
// een niet wijzigbare lijst via een int-lijst:
Integer[] values = { 2, 9, 5, 0, 3, 7, 1, 4, 8, 6 };
List<Integer> greaterThan4 = Arrays.stream(values).filter(value -> value > 4).
    collect(Collectors.toList());

// via de interface Stream zelf
Stream<String> colorStream = Arrays.stream(colors);

// intStream, longStream, DoubleStream
int[] values = { 3, 4, 6, 1 };
IntStream stream = IntStream.of(values);

// array naar stream
String[] colors = { "Red", "orange", "Yellow", "Green", "Blue" };
Stream<String> colorStream = Arrays.stream(colors);
```

```
// van collection naar stream
List<Employee> employees = new ArrayList<>();
boolean succes = Collections.addAll(employees,
new Employee("Jason", "Red", 5000, "IT"),
new Employee("Ashley", "Green", 7600, "IT"),
new Employee("Matthew", "Indigo", 3587.5, "Sales"));
Stream<Employee> employeeStream = employees.stream();
```

9 Hoofdstuk 8: String en Reguliere Expressies

9.1 Reference equality

```
    String s1 = "Hallo";
    String s2 = "Hallo";
    System.out.print(s1 == s2)
```

- Hier zal dit werken, hier krijg je `true`

```
    String s1 = new String("Hallo");
    String s2 = new String("Hallo");
    System.out.print(s1 == s2)
```

- Hier zal dit niet werken, hier krijg je `false`
- Dit kan je oplossen door `System.out.print(s1.equals(s2))` te herschrijven

9.2 Tokens - String

- Met tokens halen we de individuele woorden op van een zin

```
String zin = scanner.nextLine();
String[] tokens = zin.split(" ");
System.out.printf("Aantal elementen: %d%nTokens:%n", tokens.length);
for(String token : tokens)
    System.out.println(token);
```

9.3 Tokens - Stringbuilder

```
    String zin = scanner.nextLine();

    StringTokenizer tokens = new StringTokenizer(zin);

    System.out.printf("Aantal elementen: %d%nTokens:%n", tokens.countTokens());
    while (tokens.hasMoreTokens())
        System.out.println(tokens.nextToken());
    // end::sb[]
```

9.4 Immutiale (tip: *finale*)

- Strings zijn **immutible**, een String kan je nooit **wijzigen**
- Er zijn wel methodes die een nieuwe String maken van een al-bestaande String

```
    s1 = s1.replace("a", "e");
```

9.5 Stringbuilder

- Hier vindt je gelijkaardige methodes als in String, maar ook nieuwe
- bv. **append** en **reverse**, hierdoor kan je elk soort datatype toevoegen aan een String zonder dat het problemen geeft.

```
    Stringbuilder s1 = new StringBuilder("Hallo");
    s1.append(1).append(true).append(3.4).append("abc");
```

9.6 Reguliere expressies

- Dit gebruik je of een String voldoet aan een bepaald aantal voorwaarden

```
String regulierexpressie = "abc";
String mijnTekst = "abc";

// hier schrijven we of onze String voldoen aan de volgende voorwaarden:
// veranderen we hier iets aan de waarde, krijgen we false
System.out.println(mijnTekst.matches(regulierexpressie));
```

9.7 Regels van Reguliere expressies:

- ? : De voorafgaande token is optioneel en kan 0 of 1 keer voorkomen.
- /* : De voorafgaande token kan 0 of meer keer voorkomen.

```
String regulierexpressie = "a[^xyz]c";
String mijnTekst = "abc";
```

9.8 Characterclass - een bereik van letters en ook het omgekeerde

```
String regulierexpressie = "a[a-z]c"; // moet bevatten
String regulierexpressie = "a[^a-z]c" // niet bevatten
String regulierexpressie = "a[^a-zA-Z]c"; // niet bevatten inclusief
   hoofdletters
String regulierexpressie = "a[.]c"; // gelijk welk karakter
String regulierexpressie = "a[\D]c"; // allesbehalve een cijfer
String regulierexpressie = "a[*]c"; // 0 of meerdere karakters
String regulierexpressie = "a[?]c"; // het mag of er mag niet staan
String regulierexpressie = "a[a\\d]+c"; // 1 of meerdere keren moet dit
   groepje voorkomen
String mijnTekst = "abc";
```

9.9 Characterclass - alle speciale regex-karakters

- Dit lijkt op dat van Linux, omdat dit soort expressies komt algemeen voor in andere talen (ook buiten codetalen)

```
String regulierexpressie = "a[X?]c"; // 0 of 1 keer moet dit voorkomen
String regulierexpressie = "a[X*]c"; // 0 of oneindig keer moet dit
   voorkomen
String regulierexpressie = "a[X+]c"; // 1 tot oneindig keer moet dit
   voorkomen
String regulierexpressie = "a[X{n}]c"; // dit moet er n-keer staan
String regulierexpressie = "a[X{n, }]c"; // dit moet er minstens n-keer
   staan
String regulierexpressie = "a[X{n, m}]c"; // dit moet er minstens n-keer
   staan en maximum m-keer
```

9.10 Characterclass - Voorbeelden

- Dit lijkt op globbing van Linux, omdat dit soort expressies komt algemeen voor in andere talen (ook buiten codetalen)

```
String regulierexpressie = "[A-Z].*"; // moet beginnen met een hoofdletter
   gevolgd door een random aantal karakters
```

```
    String regulierexpressie = "[A-Z].{5,7}"; // moet beginnen met een
       hoofdletter en is 6 letters lang door een random aantal karakters
    String regulierexpressie = "([A-Z].{5,7} ){2}"; // moet beginnen met een
       hoofdletter en is 6 letters lang door een random aantal karakters
       gevolgd door een spatie en dat verwacht je 2x, maar einde is ook een
       spatie
    String regulierexpressie = "(ab)+\\d"; // a of b of ab 1 of meerdere keren
       gevolgd door een cijfer
    String regulierexpressie = "a(b|c)d"; // starten met a, b of c en eindigen
       op d

    String mijnTekst = "abc";
```

9.11 Pattern en Matcher

- Voor een grote string kun je een Pattern maken
- Hier vraag je of de tekst overeenkomt met de reguliere expressie

```
Pattern pattern = Pattern.compile(regulierexpressie);
Matcher matcher = pattern.matcher(mijnTekst);

System.out.println(matcher.find());
```

- Hier vind je alle stukjes van je String die wel voldoen aan de reguliere expressie

```
while (matcher.find)
    System.out.printf("%s gevonden die volgdoet aan de regulierexpressie",
       matcher.group())
```

9.12 Pattern en Matcher die automatisch alle mogelijke matches zoekt binnenin je zin

```
String REGEX = "a*b";
        String INPUT = "absdb";

        Pattern p = Pattern.compile(REGEX);
        Matcher m = p.matcher(INPUT); // get a matcher object
        int count = 0;

        while (m.find()) {
            System.out.println("Match " + ++count);
            System.out.println(m.group());
        }
```

9.13 Oefeningen

```
    // nemen middelste karakter
    return inhoud.charAt(inhoud.length() / 2);

    // inhoud omdraaien
    return new StringBuilder(inhoud).reverse().toString();

    // palindroom
    return inhoud.equals(geefOmgekeerdeInhoud());
```

```
    // karakters vervangen
    return inhoud.replace(Character.toLowerCase(letter1), Character.toLowerCase
       (letter2));

    // een woord splitten - hier kunnen ook reguliere expressies inzitten
    return inhoud.split(woord);
```

9.14 Oefening - een zin zijn aantal letters counten

```
    public String geefLetterRapport() {
        StringBuilder sb = new StringBuilder("Aantal klinkers: ");

        int aantalKlinkers = 0;
        int aantalMedeklinkers = 0;
        int aantalKleineletters = 0;
        int aantalHoofdletters = 0;
        int aantalCijfers = 0;
        int aantalSpecialeChars = 0;
        int aantalWoorden = 1;

        for (String letter : inhoud.split("")) {
            if (letter.matches("[aeuioAEUIO]"))
                aantalKlinkers++;
            if (letter.matches("[a-zA-Z&&[^aeiouAEIOU]]"))
                aantalMedeklinkers++;
            if (letter.matches("[a-z]"))
                aantalKleineletters++;
            if (letter.matches("[A-Z]"))
                aantalHoofdletters++;
            if (letter.matches("\\d"))
                aantalCijfers++;
            if (letter.matches("[^a-zA-Z0-9]"))
                aantalSpecialeChars++;
            if (letter.matches(" "))
                aantalWoorden++;
        }

        sb.append(aantalKlinkers).append("\n").append(String.format("Aantal
           medeklinkers: %d", aantalMedeklinkers))
                .append(String.format("%nAantal kleine letters: %d",
                   aantalKleineletters))
                .append(String.format("%nAantal hoofdletters %d",
                   aantalHoofdletters))
                .append(String.format("%nAantal cijfers: %d", aantalCijfers))
                .append(String.format("%nAantal speciale chars: %d",
                   aantalSpecialeChars))
                .append(String.format("%nAantal woorden: %d", aantalWoorden));

        return sb.toString();
    }
```

10 Hoofdstuk 9: Bestanden

10.1 URL naar het bestand:

```
// src/packagename/naamBestand
private final static String PATHNAME = "src" + File.separator + "tekst" +
    File.separator + "clients.txt";
```

10.2 OutputStream en InputStream

- Outputstream (wordt gebruikt door een formatter) + (ObjectInputStream)
  - Zet tekst vanuit een bestand naar Java
- Inputstream (wordt gebruikt door een Scanner) + (ObjectOutputStream)
  - Zet tekst vanuit Java naar een bestand

10.3 Serializable

- Bestanden die tekst halen uit een bestand moeten gebruik maken van Interface `implements Serializable`

- Om bestand te lezen en halen uit een bestand schrijf je:
  - `readObject`
  - `writeObject`

10.4 Een functie maken dat een binair bestand aanmaakt

```
public void serialiseerObjectPerObject(Collection<Speler> spelerslijst,
    String naamBestand) {
     String URL1 = "src" + File.separator + "bestanden" + File.separator +
         naamBestand;
     try (ObjectOutputStream oos = new ObjectOutputStream(Files.
         newOutputStream(Path.of(URL1)))) {
          for (Speler s : spelerslijst) {
              oos.writeObject(s);
          }
     } catch (IOException e) {
          System.err.println(e);
     }
}
```