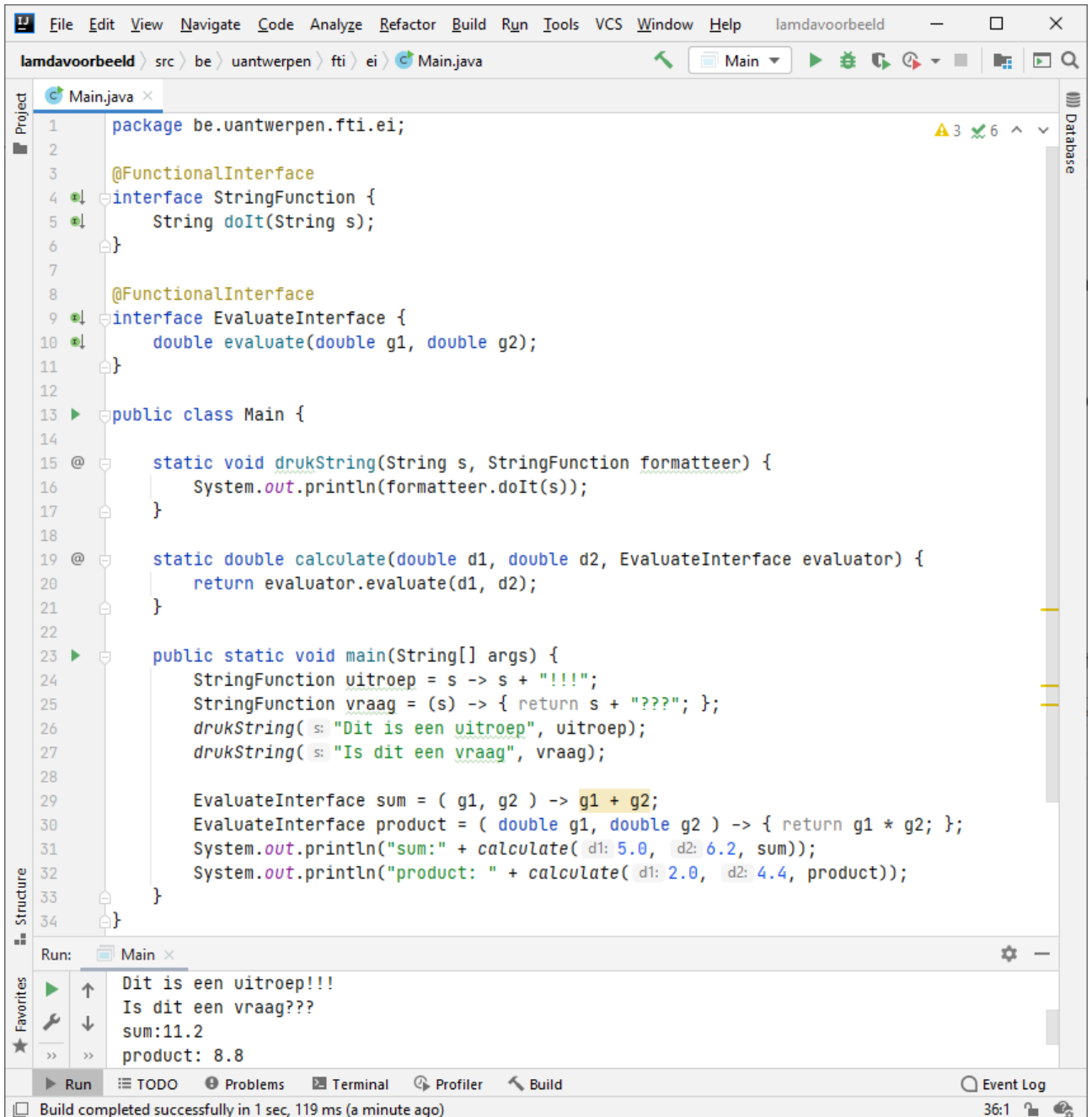


Oefeningen Java lambda's, streams API, optionals

Voorbeeld met lambda's: (je kan deze code op Blackboard terugvinden onder lambdas.txt)



```

1 package be.uantwerpen.fti.ei;
2
3 @FunctionalInterface
4 interface StringFunction {
5     String doIt(String s);
6 }
7
8 @FunctionalInterface
9 interface EvaluateInterface {
10     double evaluate(double g1, double g2);
11 }
12
13 public class Main {
14
15     @ static void drukString(String s, StringFunction formatteer) {
16         System.out.println(formatteer.doIt(s));
17     }
18
19     @ static double calculate(double d1, double d2, EvaluateInterface evaluator) {
20         return evaluator.evaluate(d1, d2);
21     }
22
23     public static void main(String[] args) {
24         StringFunction uitroep = s -> s + "!!!";
25         StringFunction vraag = (s) -> { return s + "???" };
26         drukString(s: "Dit is een uitroep", uitroep);
27         drukString(s: "Is dit een vraag", vraag);
28
29         EvaluateInterface sum = ( g1, g2 ) -> g1 + g2;
30         EvaluateInterface product = ( double g1, double g2 ) -> { return g1 * g2; };
31         System.out.println("sum:" + calculate( d1: 5.0, d2: 6.2, sum));
32         System.out.println("product: " + calculate( d1: 2.0, d2: 4.4, product));
33     }
34 }
  
```

Run: Main

```

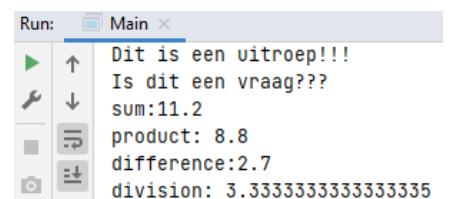
Dit is een uitroep!!!
Is dit een vraag???
sum:11.2
product: 8.8
  
```

Build completed successfully in 1 sec, 119 ms (a minute ago)

1. Voeg aan dit voorbeeld ook de EvaluateInterface toe voor:

- difference (-)
- division (/)

en test ze uit via calculate(...).



```

Run: Main
Dit is een uitroep!!!
Is dit een vraag???
sum:11.2
product: 8.8
difference:2.7
division: 3.3333333333333335
  
```

Voorbeeld Calculator met lambda's: (je kan deze code op Blackboard terugvinden onder calculator.zip)

The image displays three screenshots of an IDE (IntelliJ IDEA) showing the development of a Java calculator application using lambda expressions.

Top Screenshot: Shows the `Calculator.java` file. The class `Calculator` has private fields for `number1`, `number2`, `result`, and buttons for `sum`, `difference`, and `product`. It implements the `EvaluateInterface` with a `calculate` method. The `calculate` method uses `Double.parseDouble` to convert input strings to doubles and then uses the `evaluator` interface to perform the calculation.

Middle Screenshot: Shows the `Calculator` class with the `calculate` method and the `main` method. The `main` method creates a `Calculator` instance and sets up the GUI components (labels, buttons, and result field) with their respective positions and sizes. The `calculate` method is implemented using lambda expressions for the buttons: `sum` uses `(n1, n2) -> n1 + n2`, `difference` uses `(n1, n2) -> n1 - n2`, and `product` uses `(n1, n2) -> n1 * n2`.

Bottom Screenshot: Shows the `Main.java` file. The `Main` class has a `main` method that creates a `Calculator` instance and sets up the GUI components (labels, buttons, and result field) with their respective positions and sizes. The `calculate` method is implemented using lambda expressions for the buttons: `sum` uses `(n1, n2) -> n1 + n2`, `difference` uses `(n1, n2) -> n1 - n2`, and `product` uses `(n1, n2) -> n1 * n2`.

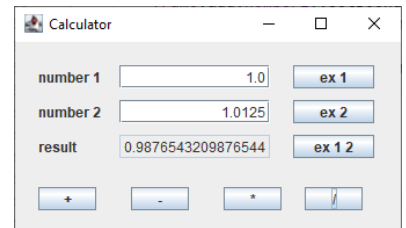
2. Pas het calculator programma aan zodat ook het quotiënt kan berekend worden (knop division en ActionListener toevoegen).

Pas de venstergrootte ook aan.

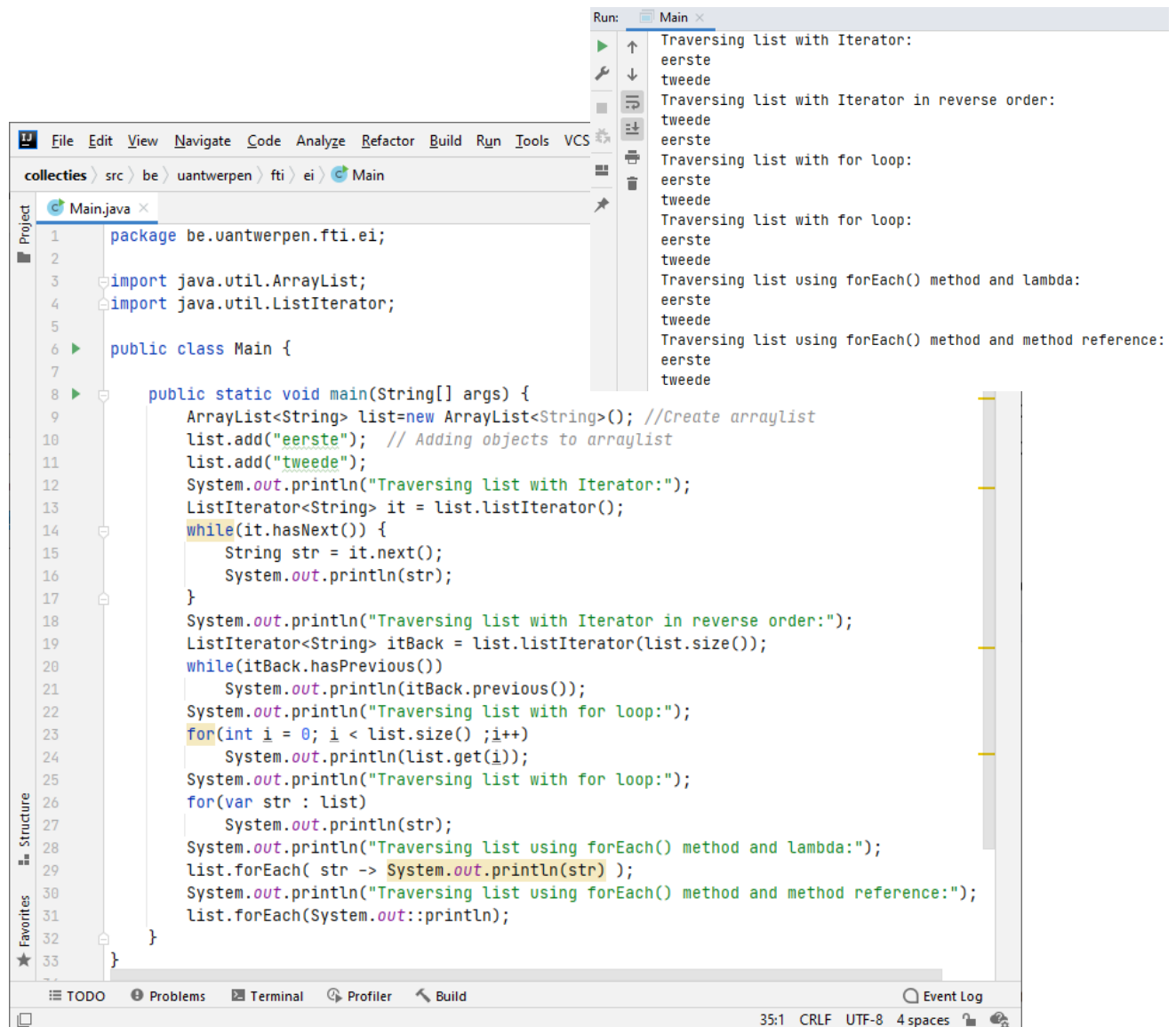
Voeg ook nog 3 knoppen toe om:

- de huidige waarde van "result" in "number1" te plaatsen
- de huidige waarde van "result" in "number2" te plaatsen
- de waarde van "number1" en "number2" om te wisselen

Zorg dat de knoppen logisch geschikt staan.



Voorbeeld itereren over ArrayList: (je kan deze code op Blackboard terugvinden onder collecties.txt)



3. Pas dit voorbeeld aan zodat de list minstens 5 voornamen bevat.

Laat de inhoud van list sorteren met de Collections sort method.

Druk in alle iteraties de namen achter elkaar af gescheiden door een spatie.

Test naast sort ook de Collections methods reverse, rotate en shuffle.

Voorbeelden van de streams API (code op Blackboard onder streamVoorbeelden.zip)

```

11 static void voorbeelden() {
12     List<String> strlijst = List.of("mia", "jan", "jef");
13     System.out.println("strlijst: "+strlijst);
14     strlijst.stream()
15         .map(String::toUpperCase)
16         .forEach(System.out::println);
17     strlijst.stream()
18         .map(s -> s.toUpperCase())
19         .forEach(s -> System.out.print(s+' '));
20     System.out.println();
21     System.out.println("Test Intstream met peek: "+
22         IntStream.of(1, 2, 3, 4)
23             .filter(e -> e > 2)
24             .peek(e -> System.out.println("Filtered value: " + e))
25             .map(e -> e * e)
26             .peek(e -> System.out.println("Mapped value: " + e))
27             .sum());
28 };
29 System.out.println("Druk de kwadraten van alle oneven getallen tussen 1 en 9:");
30 IntStream.range(1, 10)
31     .filter(i -> i % 2 != 0)
32     .map(i -> i * i)
33     .forEach(i -> System.out.print(i+" "));
34 System.out.println();
35 System.out.println("Maak een List met de kwadraten van de even getallen tussen 10 en 19:");
36 List<Integer> kwadList = IntStream.range(10, 20)
37     .filter(i -> i % 2 == 0)
38     .map(i -> i * i)
39     .mapToObj(i -> Integer.valueOf(i)) // .boxed()
40     .collect(Collectors.toList());
41 System.out.println(kwadList);
42 }

```

Run: Main

```

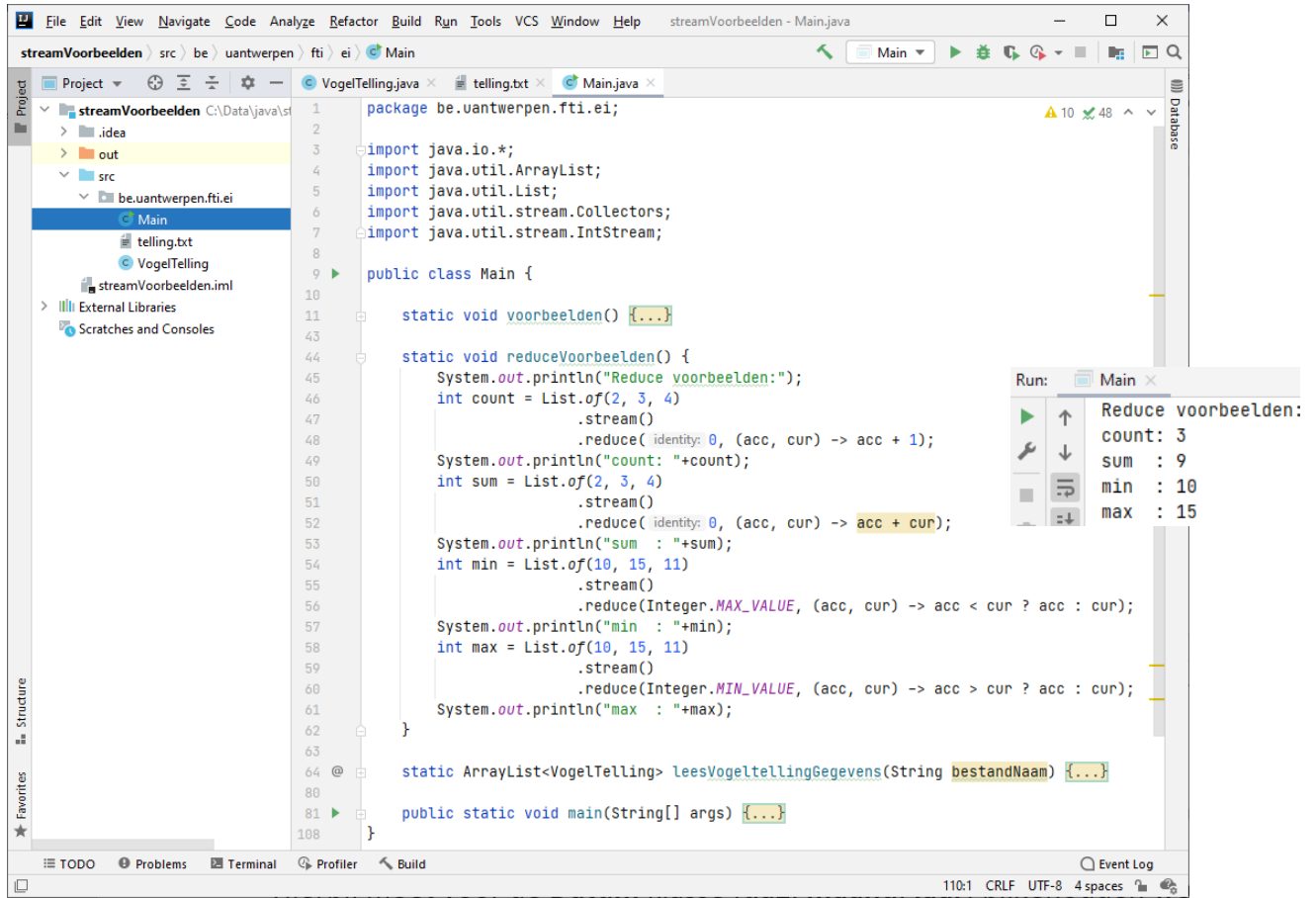
↑ strlijst: [mia, jan, jef]
↓ MIA
  JAN
  JEF
  MIA JAN JEF
  Filtered value: 3
  Mapped value: 9
  Filtered value: 4
  Mapped value: 16
  Test Intstream met peek: 25
  Druk de kwadraten van alle oneven getallen tussen 1 en 9:
  1 9 25 49 81
  Maak een List met de kwadraten van de even getallen tussen 10 en 19:
  [100, 144, 196, 256, 324]

```

Build completed successfully in 1 sec, 306 ms (a minute ago) 37:1

4. Voeg aan de functie voorbeelden() volgende stappen die enkel streams gebruiken toe:

- Maak de som van kwadraten van de even getallen tussen 1 en 50 (inclusief) (22100)
- Schrijf een functie die test of de als int doorgegeven parameter een priemgetal is (geeft als resultaat true of false terug). Dit heeft niets met streams te maken!
- Maak een list met alle priemgetallen tussen 2 en 100
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
- Maak de som van alle priemgetallen tussen 100 en 200 (3167)



```

package be.uantwerpen.fti.ei;

import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

public class Main {

    static void voorbeelden() { ... }

    static void reduceVoorbeelden() {
        System.out.println("Reduce voorbeelden:");
        int count = List.of(2, 3, 4)
            .stream()
            .reduce(0, (acc, cur) -> acc + 1);
        System.out.println("count: " + count);
        int sum = List.of(2, 3, 4)
            .stream()
            .reduce(0, (acc, cur) -> acc + cur);
        System.out.println("sum : " + sum);
        int min = List.of(10, 15, 11)
            .stream()
            .reduce(Integer.MAX_VALUE, (acc, cur) -> acc < cur ? acc : cur);
        System.out.println("min : " + min);
        int max = List.of(10, 15, 11)
            .stream()
            .reduce(Integer.MIN_VALUE, (acc, cur) -> acc > cur ? acc : cur);
        System.out.println("max : " + max);
    }

    static ArrayList<VogelTelling> leesVogeltellingGegevens(String bestandNaam) { ... }

    public static void main(String[] args) { ... }
  }

```

Run: Main x

```

Reduce voorbeelden:
count: 3
sum : 9
min : 10
max : 15
  
```

5. Voeg aan de functie `reduceVoorbeelden()` volgende stappen die enkel streams gebruiken toe:

- Tel het aantal priemgetallen tussen 100 en 150 ⁽¹⁰⁾
- Maak het product van alle oneven getallen tussen 6 en 17 (inclusief) ⁽²²⁹⁷²⁹⁵⁾
- Maak het product van alle priemgetallen tussen 5 en 20 (inclusief) ⁽¹⁶¹⁶⁶¹⁵⁾

```

package be.uantwerpen.fti.ei;

public class VogelTelling {
    private String naam;
    private int aantal;

    public VogelTelling(String naam, int aantal) {...}

    public String getNaam() { return naam; }

    public void setNaam(String naam) { this.naam = naam; }

    public int getAantal() { return aantal; }

    public void setAantal(int aantal) { this.aantal = aantal; }

    @Override
    public String toString() {
        return "VogelTelling("+"naam='"+naam+"'+'', aantal="+aantal+"')";
    }
}

```

```

static ArrayList<VogelTelling> leesVogeltellingGegevens(String bestandNaam) {
    ArrayList<VogelTelling> lijst = new ArrayList<>();
    try {
        BufferedReader reader=new BufferedReader(new FileReader(bestandNaam)); //creates a buffer
        String line = reader.readLine();
        while (line != null) {
            String telling[] = line.split( regex: " ");
            lijst.add(new VogelTelling(telling[0], Integer.parseInt(telling[1]))); // voeg vogel t
            line = reader.readLine();
        }
        reader.close(); //closes the stream and release the resources
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
    return lijst;
}

public static void main(String[] args) {
    voorbeelden();
    reduceVoorbeelden();
    ArrayList<VogelTelling> lijst = leesVogeltellingGegevens( bestandNaam: "src/be/uantwerpen/fti/ei/telling.txt");
    lijst.forEach(System.out::println);
    long aantalTellingen = lijst.stream()
        .filter(telling -> telling.getNaam().equals("koolmees"))
        .count();
    System.out.println("Aantal tellingen koolmezen: "+aantalTellingen);
    int totaalaantal = lijst.stream()
        .filter(telling -> telling.getNaam().equals("koolmees"))
        .map(telling -> telling.getAantal())
        .reduce( identity: 0, (total, aantal) -> total + aantal);
    System.out.println("Totaal aantal koolmezen: "+totaalaantal);
    int totaantal = 0;
    for (VogelTelling telling: lijst)
        if (telling.getNaam().equals("koolmees")) {
            int aantal = telling.getAantal();
            totaantal = totaantal + aantal;
        }
    System.out.println("Totaal aantal koolmezen: "+totaantal);
    int totaant = lijst.stream()
        .filter(telling -> telling.getNaam().equals("koolmees"))
        .map(telling -> telling.getAantal())
        .reduce( identity: 0, Integer::sum);
    System.out.println("Totaal aantal koolmezen: "+totaant);
}

```

telling.txt - Klabblok

```

koolmees 25
vink 38
huismus 33
koolmees 45
huismus 36
kauw 30
huismus 30
kauw 50
vink 35
koolmees 30

```

Run: Main

```

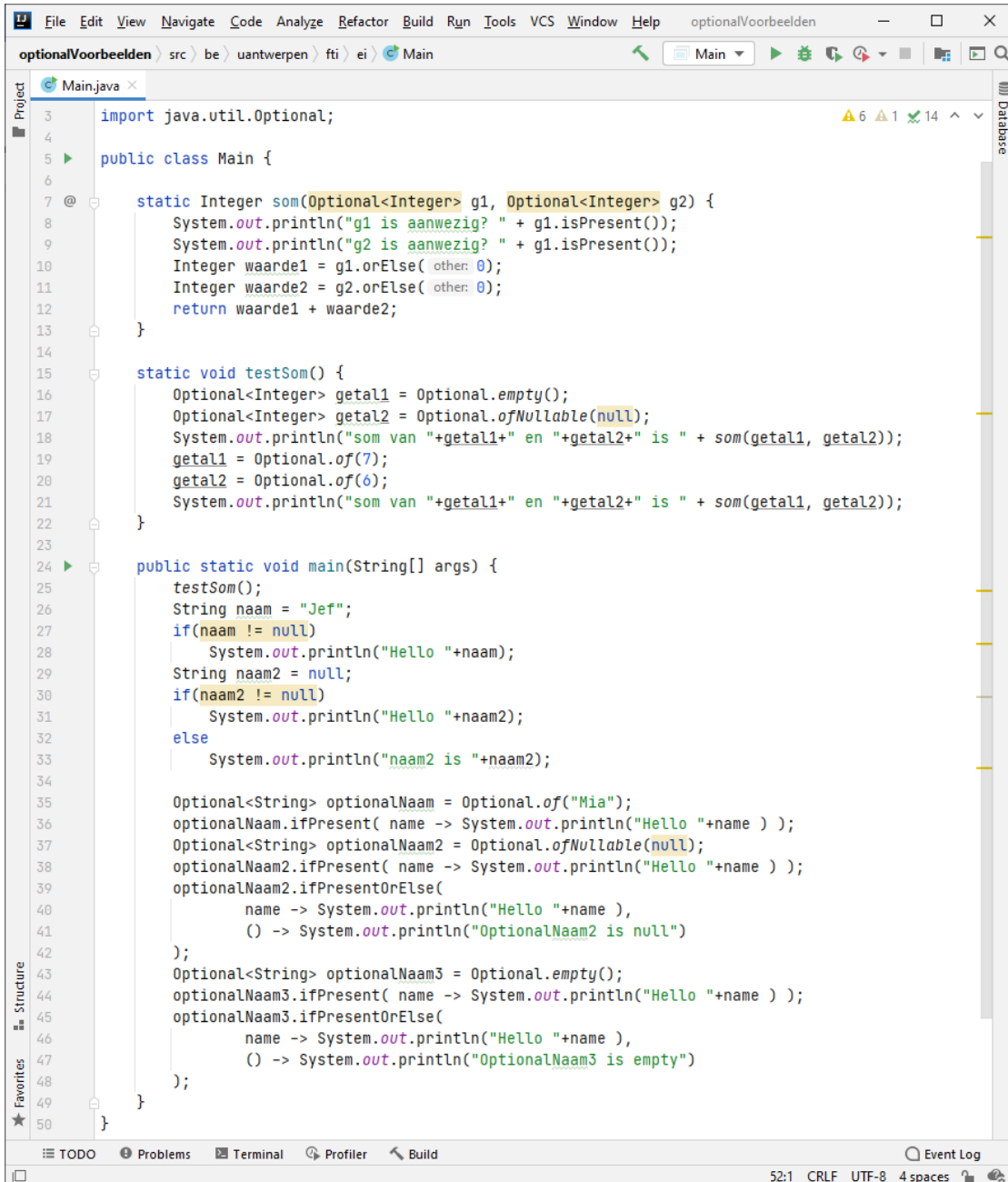
VogelTelling(naam='koolmees', aantal=25)
VogelTelling(naam='vink', aantal=38)
VogelTelling(naam='huismus', aantal=33)
VogelTelling(naam='koolmees', aantal=45)
VogelTelling(naam='huismus', aantal=36)
VogelTelling(naam='kauw', aantal=30)
VogelTelling(naam='huismus', aantal=30)
VogelTelling(naam='kauw', aantal=50)
VogelTelling(naam='vink', aantal=35)
VogelTelling(naam='koolmees', aantal=30)
Aantal tellingen koolmezen: 3
Totaal aantal koolmezen: 100
Totaal aantal koolmezen: 100
Totaal aantal koolmezen: 100

```

6. Voeg aan het voorbeeld volgende stappen die enkel streams gebruiken toe:

- Bepaal het totale aantal getelde vogels (352 duizend)
- Bepaal het totale aantal vinken (73 duizend)
- Bepaal het totale aantal kauwen en huismussen samen (één stream uitdrukking) (179 duizend)

Voorbeelden van Optionals (code op Blackboard onder optionalVoorbeelden.txt)



```

import java.util.Optional;

public class Main {

    @ static Integer som(Optional<Integer> g1, Optional<Integer> g2) {
        System.out.println("g1 is aanwezig? " + g1.isPresent());
        System.out.println("g2 is aanwezig? " + g1.isPresent());
        Integer waarde1 = g1.orElse( other: 0);
        Integer waarde2 = g2.orElse( other: 0);
        return waarde1 + waarde2;
    }

    static void testSom() {
        Optional<Integer> getal1 = Optional.empty();
        Optional<Integer> getal2 = Optional.ofNullable(null);
        System.out.println("som van "+getal1+" en "+getal2+" is " + som(getal1, getal2));
        getal1 = Optional.of(7);
        getal2 = Optional.of(6);
        System.out.println("som van "+getal1+" en "+getal2+" is " + som(getal1, getal2));
    }

    public static void main(String[] args) {
        testSom();
        String naam = "Jef";
        if(naam != null)
            System.out.println("Hello "+naam);
        String naam2 = null;
        if(naam2 != null)
            System.out.println("Hello "+naam2);
        else
            System.out.println("naam2 is "+naam2);

        Optional<String> optionalNaam = Optional.of("Mia");
        optionalNaam.ifPresent( name -> System.out.println("Hello "+name ) );
        Optional<String> optionalNaam2 = Optional.ofNullable(null);
        optionalNaam2.ifPresent( name -> System.out.println("Hello "+name ) );
        optionalNaam2.ifPresentOrElse(
            name -> System.out.println("Hello "+name ),
            () -> System.out.println("OptionalNaam2 is null")
        );
        Optional<String> optionalNaam3 = Optional.empty();
        optionalNaam3.ifPresent( name -> System.out.println("Hello "+name ) );
        optionalNaam3.ifPresentOrElse(
            name -> System.out.println("Hello "+name ),
            () -> System.out.println("OptionalNaam3 is empty")
        );
    }
}

```

Run: Main

```

g1 is aanwezig? false
g2 is aanwezig? false
som van Optional.empty en Optional.empty is 0
g1 is aanwezig? true
g2 is aanwezig? true
som van Optional[7] en Optional[6] is 13
Hello Jef
naam2 is null
Hello Mia
OptionalNaam2 is null
OptionalNaam3 is empty

```

Run: Main

```

Priemgetallen(2..50): [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
Priemgetallen(32..36): Het interval bevat geen priemgetallen!

```

6. Schrijf een functie die de priemgetallen tussen de als parameters doorgegeven van ... tot als list teruggeeft. Maak van deze returnwaarde een Optional die ofwel de list teruggeeft ofwel empty() teruggeeft als er geen priemgetallen zijn in het bereik van ... tot. Maak bij de oproep gebruik van ifPresentOrElse() om het resultaat af te drukken.

7. Schrijf een Java programma dat een class gebruikt om een datum vast te leggen.
(zie opgave 2 van reeks 1)

Hierbij moet voor de **Datum** klasse (**dag, maand, jaar**) bijgehouden worden (3 x **int**).

Verder moeten methods voorzien worden om:

- de gegevens te initialiseren (constructors)
- de gegevens te wijzigen (set)
- de gegevens individueel op te vragen als getal (dag, maand en jaar)
- de gegevens terug te geven als string (toString())
- naar de volgende dag te gaan (method next of volgende)
- de gegevens te controleren op correctheid (vb. dagen per maand) Let op voor schrikkeljaren
- tijdstippen te vergelijken (interface Comparable, method compareTo)
- de huidige dag op halen in default constructor
- gebruik **exceptions** om een foutieve datum op te vangen in de constructor en de set methods. Creëer hiervoor een exception class “OngeldigeDatum” (InvalidDate).

Het hoofdprogramma moet een demonstratie geven van het gebruik van alle methods.

Toevoeging reeks 2:

Maak een ArrayList met minstens 10 Datum objecten.

Formuleer nu stream uitdrukkingen met volgend resultaat:

- Tel het aantal datums met als jaar 2023
- Tel het aantal datums met als maand 12
- Maak een list met alle datums met als jaar 2023
- Maak een list met alle datums waarvan het jaar tussen 2015 en 2021 ligt
- Maak een list met alle datums waarvan de maand gelijk is aan 7 of 8

Druk alles netjes af.

8. Schrijf een Java programma voor volgende opgave: (zie opgave 3 van reeks 1)

Maak een class om een **Persoon** voor te stellen met als data members (private):

- string naam;
- string gebdat; // geboortedatum "dd/mm/jjjj"
- string adres;

Voorzie alle nodige methods. (get, set, constructors (default, init) , toString)

Maak een tweede class **Student** afgeleid van Persoon met als extra data members (private):

- string klas;
- double punten; // resultaat in %

Voorzie alle nodige functies. (get, set, constructors (default, init) , toString)

Maak een derde class **Kotstudent** afgeleid (public) van Student met als extra data member (private):

- string kotadres;

Voorzie alle nodige functies. (get, set, constructors (default, init) , toString)

Elke class krijgt een eigen functie om zijn gegevens om te zetten naar string (toString).

Zorg ervoor dat in alle gevallen de juiste gegevens netjes op het scherm worden weergegeven.

Demonstreer het gebruik van deze drie classes in het hoofdprogramma.

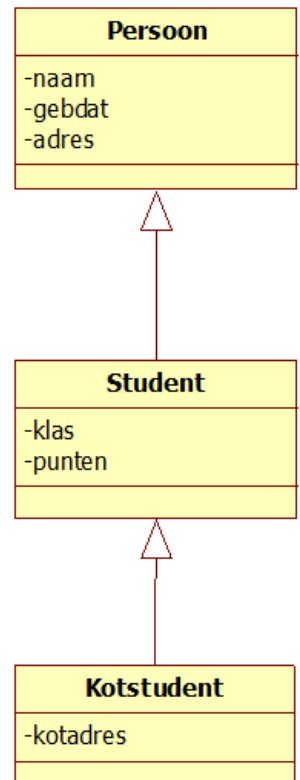
Begin met persoon.

Voeg daarna student toe.

Voeg kotstudent maar toe als de eerste twee klassen in orde zijn en werken.

Het hoofdprogramma moet een demonstratie geven van het gebruik van alle methods.

Gebruik hierbij de werking van het polymorfiemechanisme bij het implementeren van toString.



Toevoeging reeks 2:

Maak een ArrayList met minstens 5 Student objecten.

Formuleer nu stream uitdrukkingen met volgend resultaat:

- Tel het aantal studenten met punten hoger of gelijk aan 85
- Tel het aantal studenten met punten hoger of gelijk aan 50
- Maak een list met alle studenten met punten hoger of gelijk aan 50
- Maak een list met alle studenten met punten kleiner dan 50
- Maak een list met alle studenten van klas 2EI met punten hoger of gelijk aan 75

Druk alles netjes af.

9. Bedenk zelf nog enkele originele opgaven waarbij je lambda's, streams en optionals gebruikt en werk ze uit.