

Labsessie week 7

Lussen en file-input

Theorie

Theorie die je vóór deze labosessie moet verwerkt hebben, heb je gezien in week 3: collections 1 en week 5: fouten en data-IO.

Doel

In deze labosessie bekijken we twee manieren om een blok code herhaaldelijk uit te voeren: `while`- en `for`-lussen. Bovendien bekijken we een manier om informatie uit te lezen uit een tekstbestand.

Formaat

Elke labosessie is onderverdeeld in drie verschillende delen: startoefeningen, labo-oefeningen en thuisoefeningen. **Startoefeningen** zijn oefeningen waarvan we verwachten dat je ze thuis oplost, vóór de labosessie. Op die manier geraak je vertrouwd met de materie en check je of je de theorie van de hoorcolleges goed begrepen en verwerkt hebt. **Labo-oefeningen** zijn oefeningen die tijdens de labosessie worden behandeld. **Thuisoefeningen** zijn oefeningen die dieper ingaan op de materie. Ze zijn optioneel voor wie sneller werkt, of als extra oefeningmateriaal bij het studeren/oefenen thuis. Neem contact op met jouw labo-assistent voor feedback op deze oefeningen.

Leerdoelstellingen

Na deze sessie moet je in staat zijn om de volgende concepten te begrijpen en te implementeren:

- While- en forlussen schrijven om code herhaaldelijk te laten uitvoeren.
- Verschillen en gelijkenissen kunnen inzien tussen de twee.
- Data uit een tekstbestand uitlezen en gebruiken in een programma.

Startoefeningen

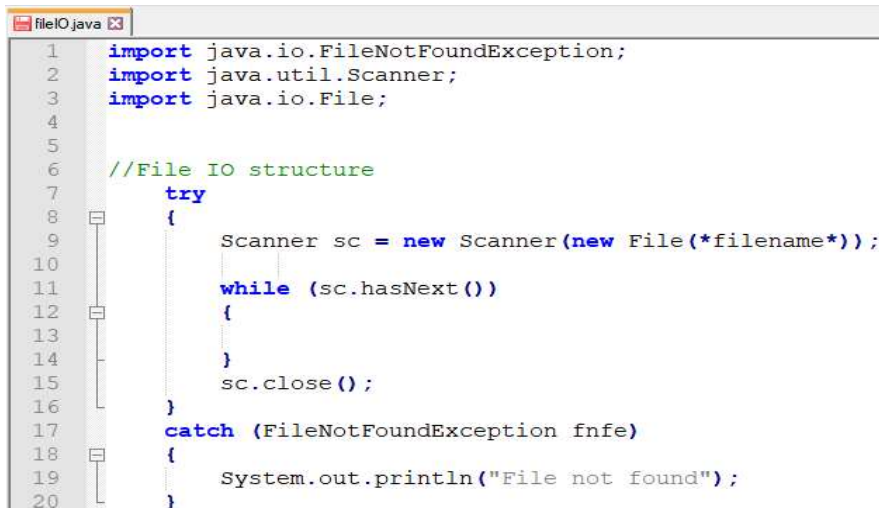
Open het project "StarterExercises" op Toledo en bekijk de volgende klassen.

Oefening 1: Lussen

1. Maak een object aan van de klasse Loops en pas de twee gegeven methodes toe op dit object. Open de klasse Loops en probeer te code te begrijpen met behulp van de commentaren.
2. Schrijf, met behulp van een **for-lus**, een methode printNumbersToFor(int upTo), die de getallen tot **en met** de gegeven parameter print.
3. Schrijf, met behulp van een **while-lus**, printEvenNumbersToWhile(int upTo), die de **even** getallen tot (**niet met**) de gegeven parameter print.
4. Schrijf, met behulp van een **for-lus**, printEvenOrOddToFor(int upTo, boolean even), die, afhankelijk van de boolean even, de even of oneven getallen print, tot (**niet met**) de gegeven parameter. printEvenOrOddToFor(7, true) print dus wél tot en met 6.

Oefening 2: File I/O

File I/O omvat het gebruiken van code om van en naar een bestand te lezen en te schrijven (meestal een tekstbestand). Voor deze oefeningen (inclusief op het examen) krijg je **ten minste** het volgende codefragment:



```
1 import java.io.FileNotFoundException;
2 import java.util.Scanner;
3 import java.io.File;
4
5
6 //File IO structure
7 try
8 {
9     Scanner sc = new Scanner(new File(*filename*));
10
11     while (sc.hasNext())
12     {
13
14     }
15     sc.close();
16 }
17 catch (FileNotFoundException fnfe)
18 {
19     System.out.println("File not found");
20 }
```

Laten we dit in afzonderlijke delen bekijken:

- Lijnen 1 tot 3 bevatten importeerinstructies. Deze lijnen code laden de vereiste bibliotheken in je project. Waar dit op neerkomt is dat klassen en methodes die geschreven zijn door andere Javaprogrammeurs worden toegevoegd zodat jij ze kan gebruiken. Zet deze statements altijd bovenaan je klasse.
- Lijnen 7, 17-20 bevatten een try-catch-instructie. Try-catch-instructies worden gebruikt om je code te beschermen tegen lijnen code die een foutmelding kunnen geven. In dit geval wordt de mogelijke fout veroorzaakt door het geven van een bestandsnaam van een bestand dat niet bestaat (vandaar de naam "FileNotFoundException" op lijn 17). In plaats van een foutmelding te geven en je programma te stoppen zal een try-catch-blok de lijnen code in het catch-blok

(lijnen 17-20) uitvoeren als de fout zich voordoet en zal je code zoals normaal verder uitgevoerd worden.

- Lijn 9 creëert een nieuwe instantie van de Scanner-klasse (waar we de klasse van geïmporteerd hebben op lijn 2). De parameter die gegeven wordt aan de constructor van het Scanner-object is een File-object (geïmporteerd op lijn 3) waarvan de constructorparameter de locatie is van het bestand, gegeven als string. De verbinding tussen ons Javaprogramma en het tekstbestand wordt gesloten op lijn 15.
- Lijnen 11-14 bevatten de while-lus waarin we de inhoud van het bestand kunnen gebruiken. Je kan de Javadocdocumentatie gebruiken om op te zoeken wat "sc.hasNext()" doet. Voor deze keer is de samenvatting van de methode hieronder gegeven. Het komt erop neer dat deze methode een boolean teruggeeft die "true" is als er nog een token in het tekstbestand aanwezig is, en "false" is als het einde van het bestand bereikt is.

Method Summary

Methods	
Modifier and Type	Method and Description
boolean	<code>hasNext()</code> Returns true if this scanner has another token in its input.

Om de Scanner-klasse correct te kunnen gebruiken moeten we de Javadocdocumentatie gebruiken. Hoewel je deze ook online kan vinden op <https://docs.oracle.com/en/java/javase/11/docs/api/>, is het aangeraden om het zip-bestand van Toledo te downloaden, onder oefeningendocumenten, "docs.zip". Op het examen wordt de documentatie in deze vorm voorzien. Je kan toegang krijgen tot de Javadocdocumentatie door het bestand te downloaden naar een gekende locatie op je computer, het te unzippen en door op "index.html" te dubbelklikken in de api-map. Via het menu bovenaan kan je de lijst van alle klassen terugvinden. Je kan de Scanner-klasse vinden door hierdoor te scrollen of via de zoekfunctionaliteit van je browser (ctrl+f). De documentatie van klassen start met een samenvatting van wat de klasse doet, een beschrijving van de constructor en een opsomming van de methodes. Het is essentieel om deze opsomming door te nemen om de juiste methodes te selecteren om te gebruiken in deze oefeningen.

1. Maak een object aan van de klasse FileIO en pas de methode toe op dit object. Open de klasse FileIO en probeer de code te begrijpen met behulp van de commentaren.
2. Implementeer de methode "findGivenInteger()", die momenteel in commentaar staat. Deze functie geeft het lijnnummer waar een gegeven getal in het bestand staat. Uiteraard start je te tellen vanaf 0. Geef -1 terug als dit getal niet aanwezig is in het bestand.

Labo-oefeningen

Oefening 1: Printen met lussen

Open het "LoopExercise"-project op Toledo en vervul de methoden in de klasse LoopExercisePrint. Tip1: Als methodes te groot worden kan je ze opdelen in verscheidene methodes en deze methodes oproepen in de originele methode. Tip2: Gebruik de debugger om makkelijker fouten in je logica te detecteren.

Methode 1. Print de vermenigvuldigingstabel van een gegeven getal in het volgende formaat. Tip: gebruik "\t" (=tab) als speciaal karakter in je printinstructie om de kolommen uit elkaar te houden. Een voorbeeld voor het getal 5:

5:	5	10	15	20	25	30	35	40	45	50
----	---	----	----	----	----	----	----	----	----	----

Methode 2. Print de vermenigvuldigingstabel van één tot een gegeven getal. Tip: gebruik "\n" of println() om een nieuwe lijn te beginnen. Als voorbeeld voor het getal 3:

1:	1	2	3	4	5	6	7	8	9	10
2:	2	4	6	8	10	12	14	16	18	20
3:	3	6	9	12	15	18	21	24	27	30

Methode3. **[ga eerst door met de volgende oefeningen en kom later op deze oefening terug!]** Schrijf een lus die de getallen van 40 tot 200 print met telkens 10 getallen per lijn. Telkens een getal een meervoud van 4 is, wordt "****" geprint in plaats van het getal. Telkens een getal een meervoud van 9 is, wordt "+++" geprint in plaats van het getal. Als een getal een meervoud is van meerdere van deze getallen wordt een chronologische concatenatie geprint. Als voorbeeld: voor 72, dat een meervoud is van 4 en 9, wordt "****+++" geprint.

Oefening 2: Lussen in methodes met return

Nu vervul de code in de klasse LoopExerciseReturn. Gebruik de testklasse om je oplossing te controleren.

Methode1. Bereken hoeveel keer het gegeven karakter voorkomt in een gegeven string. Zoek zelf op in de java-API hoe je de lengte van een string kan bepalen. Idem hoe je een karakter op een gegeven positie kan inlezen.

Methode2. **[ga eerst door met de volgende oefeningen en kom later op deze oefening terug!]** Bepaal of het gegeven karakter voorkomt op een oneven positie in een gegeven string. Waarom is het (iets) beter hier een while-lus te gebruiken?

Methode3. Bereken het aantal stappen dat nodig is om 1 te bereiken als je het volgende algoritme gebruikt. Je start met een gegeven natuurlijk getal en voert volgende operaties uit:

- Het getal wordt $n/2$ als het getal even was.
- Het getal wordt $3n+1$ als het getal oneven was.

Herhaal dit algoritme met het resulterende getal. Men vermoedt dat je op die manier altijd tot bij 1 geraakt, in een eindig aantal stappen! Geef het aantal stappen dat hiervoor nodig was, terug. In het geval van ongeldige situaties (nul, negatief getal...) geef je -1 terug. Waarom gebruik je hier best een while-lus?

Oefening 3: File I/O

Open het "FileIOExercise"-project op Toledo en vervulledig elke methode. Gebruik de testklasse om je oplossing te controleren. Tip: Gebruik de debugger om makkelijker fouten in je logica te detecteren.

Methode 1. Bepaal het aantal keer dat een gegeven integer voorkomt in een tekstbestand (gegeven als parameter) en geef deze waarde terug. Return -1 bij een niet-gevonden bestand.

Methode 2. **[ga eerst door met de volgende oefeningen en kom later op deze oefening terug!]**

Bepaal het product van de eerste 10 getallen in een tekstbestand (gegeven als parameter) en geef deze waarde terug. Tip: je zal een apart tellertje moeten bijhouden om te kijken hoeveel lijnen je al uitgelezen hebt. Return -1 bij een niet-gevonden bestand.

Methode 3. Bepaal het aantal keer dat een gegeven karakter voorkomt in een tekstbestand (gegeven als parameter) en geef deze waarde terug. Return -1 bij een niet-gevonden bestand.

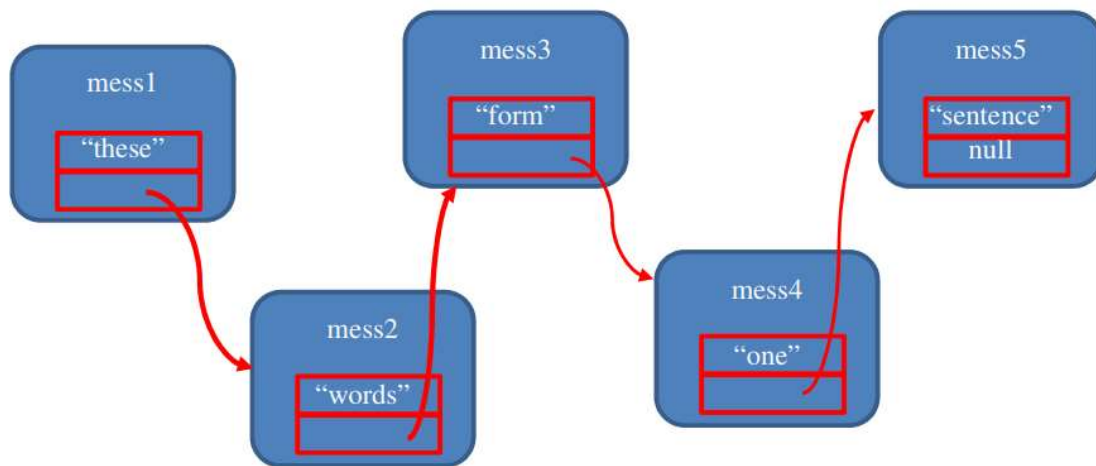
Thuisoefeningen

Oefening 1: Harmonische reeks

De harmonische reeks $\sum_{n=1}^{\infty} \frac{1}{n}$ divergeert. Schrijf een methode `termsNeeded(int reach)` die berekent hoeveel termen van de harmonische reeks je nodig hebt om een gegeven getal *reach* te bereiken.

Oefening 2: Referenties [Uitdaging]

Maak een klasse "Message" die naast het attribuut "word" van het type String ook een attribuut "next" bevat, dat een referentie is naar een ander Message-object. Hierdoor wordt het mogelijk om een lijst van opeenvolgende Message-objecten te maken. Een null-referentie voor de opvolger wil zeggen dat het bericht ten einde is. Beide attributen worden toegewezen in de constructor. Implementeer een test die de volgende situatie genereert:



Voeg de volgende functionaliteit toe (naast de benodigde setters en getters):

1. `+countFollowers() : int`
Deze methode geeft het aantal objecten terug die volgen op het huidige object. Als voorbeeld: deze methode zou 2 moeten teruggeven voor `mess3` en 0 voor `mess5`.
2. `+concatenateMessages(int numberOfFollowers, boolean decoration) : String`
Deze method concateneert alle woorden van een aantal (`numberOfFollower`) objecten startende van het huidige object tot één string en geeft deze terug. Tussen elk woord wordt een spatie toegevoegd. Als `decoration true` is, wordt een punt toegevoegd aan het einde van de string. Als `numberOfFollowers` gelijk is aan -1 wordt de lijst tot op het einde doorzocht. Als voorbeeld: `mess1.concatenateMessages(2, false)` resulteert in "these words form". `mess1.concatenateMessages(-1, true)` resulteert in "these words form one sentence."
3. `+capitalize() : void`
Verander alle letters in het woord naar hoofdletters.
4. `+totalNumberOfCharacters() : int`
Deze method geeft het aantal karakters terug waaruit de String bestaat die het resultaat is van `concatenateMessages(-1, false)`.