# WEEK 4

## ONDERWERPEN

- TDD using assertions
- handling exceptions
- file i/o
- i/o with JSON
- networking
- profiling
- build tools

## OPMERKING

Alle opgave nummers verwijzen naar Liang, **10e editie**. Om verwarring te voorkomen zijn de opgaven uit Liang hier voor een deel overgenomen, maar dit is niet altijd compleet. Voor de volledige versie moet je het boek van Liang raadplegen.

## ASSERTIONS AND JUNIT

a) On Blackboard you can find the file "start code week4.rar". In src\main\java\com\hanze\gradle you find the file Person.java. Now add a PersonTest.java in src\test\java\com\hanze\gradle\test with the following class :

```
public class PersonTest {
    @Test
    public void test() {
        Person person = new Person("Jeremy");
        assertEquals(person.getName(), "Jeremy");
    }
}
```

b) What does the annotation @Test do ?

c) Using the JUnit test framework, show that Person.java passes this test. Of course, you may add some more tests.

## OPGAVEN EXCEPTIONS

**Hoofdstuk 12 pag. 455 Review vragen.**

12.5 What is the output of the following code?
12.6 Show the output of the following code.
12.7 Describe the Java Throwable class, its subclasses, and the types of exceptions.
12.8 What RuntimeException will the following programs throw, if any?
12.13 Suppose that statement2 causes an exception in the following try-catch block:
12.16 What is displayed when the following program is run?
12.21 Suppose that statement2 causes an exception in the following statement:
12.23 Suppose that statement2 causes an exception in the following statement:

**Hoofdstuk 12 pag. 489 Programming Exercises.**

12.6 (*NumberFormatException*) Listing 6.8 Hex2Dec.java (see Blackboard) implements the hex2Dec(String hexString) method, which converts a hex string into a decimal number. Implement the hex2Dec method to throw a NumberFormatException if the string is not a hex string.

## OPGAVEN FILE I/O

**Hoofdstuk 12 pag. 482 Review vragen.**

12.33 How do you create a Scanner to read data from a file? What is the reason to define throws Exception in the main method in Listing 12.15, ReadData.java? What would happen if the close() method were not invoked in Listing 12.15?

12.35 Is the line separator the same on all platforms? What is the line separator on Windows?

**Hoofdstuk 12 pag. 491 Programming Exercises.**

12.14 *(Process scores in a text file)* Suppose that a text file contains an unspecified number of scores separated by blanks (see Exercise12_14.txt on Blackboard). Write a program that prompts the user to enter the file, reads the scores from the file, and displays their total and average.

12.25 (*Process large dataset*) A university posts its employees' salaries at http://cs.armstrong.edu/liang/data/Salary.txt. Each line in the file consists of a faculty member's first name, last name, rank, and salary (see Programming Exercise 12.24). Write a program to display the total salary for assistant professors, associate professors, full professors, and all faculty, respectively, and display the average salary for assistant professors, associate professors, full professors, and all faculty, respectively.
Hint : you can use java.net.UR and java.util.Scanner.

**Hoofdstuk 17 Programming Exercise.**

Rewrite listing 17.8 on page 699 using Files.newByteChannel().

## OPGAVE JSON

In de sheets van week 4.1 is een programma JsonSimpleExample.java dat een file book.json parsed (zie source file op Backboard). Maak een programma dat het omgekeerde doet, het aanmaken van de file book.json met behulp van de JSON.simple library.

## OPGAVE NETWORKING

**Hoofdstuk 31 pag. 1171 Programming Exercises.**

31.9 (*Chat*) Write a program that enables two users to chat. Implement one user as the server (Figure 31.21a) and the other as the client (Figure 31.21b). The server has two text areas: one for entering text and the other (non-editable) for displaying text received from the client. When the user presses the *Enter* key, the current line is sent to the client. The client has two text areas: one (non-editable) for displaying text from the server and the other for entering text. When the user presses the *Enter* key, the current line is sent to the server.

## OPGAVE HPROF

Informatietheorie is een wiskundig vakgebied dat zich bezig houdt met het zo efficiënt en betrouwbaar mogelijk opslaan of overdragen van informatie. Een belangrijk onderwerp binnen informatietheorie is broncodering. Bij broncodering worden coderingstechnieken toegepast zodat overbodige informatie (z.g.n. redundante informatie) achterwege blijft, wat geheugen- of opslagruimte spaart. De informatie kan op deze manier worden gecomprimeerd. Bij broncodering kan de bron bestaan uit tekst, data, audio of video.

Bij audio en video kan er bovendien informatie weg worden gelaten die een luisteraar of kijker toch niet kan waarnemen (bijvoorbeeld bij tegelijk een zacht en een hard geluid wordt het zachte geluid niet waargenomen). Er kan dan gebruik worden gemaakt van niet-omkeerbare ('lossy') compressie, zoals bijvoorbeeld bij mp3. Het oorspronkelijke bronbestand kan dan niet meer terug worden ge-decodeerd. Bij tekst en databestanden zal de compressie wel omkeerbaar ('lossless') moeten zijn. Je wilt immers geen tekst kwijtraken. Bekende omkeerbare compressie technieken zijn Huffman codering en Lempel Ziv Welch (1984) codering. LZW wordt o.a. gebruikt in jpg.

Er is ook een interessant verband tussen machine learning (regressie en classifiers) en data compressie. Bij beide gaat het immers om systemen die toekomst proberen te voorspellen op basis van data uit het verleden.

Het LZW-algoritme gebruikt een zogenaamd 'woordenboek' om het bestand te comprimeren. In plaats de tekens afzonderlijk te coderen houdt dit algoritme een lijst bij (het 'woordenboek') van reeksen van tekens (= substrings). Het idee is dat substrings die vaker voorkomen kunnen worden vervangen door een korte binaire code. Deze code vertelt de de-compressor hoeveel symbolen er in de bron zaten en welke dit zijn.

Relevante URL's :

https://en.wikipedia.org/wiki/Data_compression
http://en.wikipedia.org/wiki/Lempel-Ziv-Welch
http://www.cs.duke.edu/csed/curious/compression/lzw.html

Op Blackboard is een programma LZW.java te vinden. Pas de regel '`List<Integer> compressed =`' aan zodat een grote tekstfile wordt ingelezen. Je kan hiervoor astronaut.txt gebruiken, zie Blackboard.

Analyseer met HPROF welke methoden/statements de meeste CPU capaciteit nemen, en analyseer welke objecten de meeste ruimte op de heap vragen. Onderbouw je bevindingen met delen uit java.hprof.txt.

## OPGAVEN GRADLE

Met Gradle kun je ook automatisch applicaties testen. Gradle heeft build-in ondersteuning voor o.a. het JUnit test framwork. In de opgave "assertions and Junit" heb je de klasse TestPerson gemaakt. Plaats Testperson.java onder 'src/test/java'. De locatie van je testcode kun je aanpassen door het configureren van de 'test' source set. Gradle zal straks automatisch de productie- en testcode compileren en uitvoeren.

a) Wat nog handmatig moet gebeuren is het configureren van de test dependencies. Voeg een JUnit dependancy toe aan de 'testCompile' configuration.
b) Voer de test uit met `$ gradle test`.
c) Pas de sourcefile PeronTest.java aan zodat de assertion niet meer geldt en de test faalt. Gradle zal melden: Execution failed for task ':test'. Kijk in de 'build/reports' directory naar het testrapport.