GROEIPROJECT
MODULE 4: TESTEN EN LOGGING

1. Voorbereiding

1.1. Open het groeiproject en maak daarin een vierde module: `4_testen`

1.2. Kopieer van module `1_herhaling` de hele src map (met o.a. de oorspronkelijke basis- en multiklasse) naar de nieuwe module.

1.3. Maak in de directory van je project een map met de naam `lib` en zet daarin de benodigde jars: `hamcrest-core-1.3` en `junit-4.12`. Je vindt ze ook op Blackboard.
Gebruik de menukeuze "*File* > *Project Properties* > *Project Settings* > *Libraries*" om naar deze jars te verwijzen.

2. Testen met JUnit

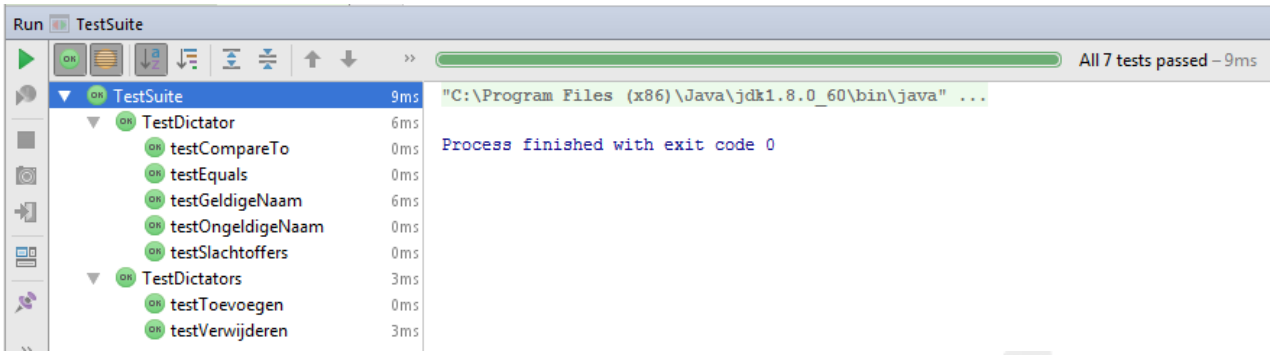
2.1. Maak **NAAST** de `src`-folder een nieuwe folder met de naam `test`. Ga als volgt te werk:
   a. Ga naar het Project-venster en positioneer de cursor op de module `4_testen`
   b. Gebruik de menukeuze: "*File* > *New* > *Directory*"
   c. Kies als naam: `test`
   d. Gebruik rechter muisknop op `test` en kies: "*Mark Directory As* > *Test Sources Root*"
   e. In IntelliJ zal de folder en alles wat erin zit nu groen worden gekleurd.

2.2. Maak een testklasse voor je basisklasse; in ons geval `TestDictator`:
   a. Ga in de broncode van de basisklasse op de klassenaam staan (bij ons Dictator),druk ALT+ENTER en selecteer *Create Test*
   b. Kies JUnit 4 als Testing Library (klik eventueel "fix" en kies JUnit 4 from IntelliJ distribution als te installeren bibliotheek).
   Verander de naam van de klasse naar `TestDictator`.
   Verifieer dat `TestDictator` in de `test` map in dezelfde package als `Dictator` aangemaakt is.
   Laat de andere opties voorlopig open.
   c. Voorzie in de testklasse minstens twee attributen met objecten van je basisklasse.
   d. Druk ALT+INS in de test klasse en kies *SetUp Method*. Gebruik de gegenereerde `@Before` methode om de attributen te instantiëren.
   e. Schrijf een methode `testEquals`, waarin je de `equals` methode van je basisklasse uittest. Je mag een positief (2 gelijke objecten) en een negatief (twee ongelijke objecten) geval in dezelfde methode testen.
   f. Schrijf een methode `testOngeldig` waarin je een ongeldige waarde aan een setter van de basisklasse meegeeft en dus een `IllegalArgumentException` verwacht (zie: Groeiproject_1 > Opdracht 2.2)

g. Schrijf een methode `testGeldig` waarin je via een setter een correcte waarde instelt en er dus GEEN `IllegalArgumentException` mag optreden.
h. Schrijf een methode `testCompareTo` waarin je de sorteervolgorde uittest
i. Schrijf ook een testmethode waarin je `assertEquals` gebruikt om de gelijkheid van 2 double-waarden te vergelijken met als laatste parameter een delta-waarde als tolerantie.
j. Controleer of alle assert-methoden die je gebruikt hebt ook telkens een passende foutmelding bevatten.
k. Run en controleer of je groen licht krijgt voor alle testen.

2.3. Maak een testklasse voor je multiklasse; in ons geval `TestDictators`:.
a. Voorzie opnieuw de nodige attributen en een `@Before` methode om die in te stellen
b. Voorzie een methode om het toevoegen uit te testen
c. Voorzie een methode om het verwijderen uit te proberen
d. Run en controleer of je groen licht krijgt voor alle testen

2.4. Maak een testklasse met de naam `TestSuite` en combineer daar de beide testklassen.
Run en controleer de output:



➔ Controleer ook eens via de "Code coverage" ( ) en controleer voor hoeveel % jouw code in basis- en multiklasse gecovered is.

2.5. Maak tenslotte een klasse `TestRunner` met een main-methode, waarin je de `TestSuite` via code opstart en het resultaat via een `Result`-object onderzoekt. Druk af: de failures, succesful, het aantal testcases en de benodigde tijd.

2.6. Mogelijke afdruk:
```
Failures:
Successful: true
Aantal testcases: 7
Tijd: 9 millisec
```

2.7. Mogelijke afdruk (met fout):
```
Failures:
testCompareTo(TestDictator): 2 dictators met dezelfde naam moeten compareTo = 0 geven
expected:<0> but was:<1>
Successful: false
Aantal testcases: 7
Tijd: 13 millisec
```

3. LOGGING

3.1. Pas in de basisklasse de getters die een Object teruggeven aan, zodat de methode een default waarde (voorbeeld "Onbekend") teruggeeft indien het attribuut null is.

3.2. Voorzie in de basisklasse een extra `Logger`-attribuut. In alle setters waar je een IllegalArgumentException gooit als de meegegeven parameter foutief is, ga je daarnaast ook een logging doen met niveau SEVERE. De boodschap moet (naast tekst) ook de foute waarde en de naam van het object bevatten.
Bijvoorbeeld: `"Geboortedatum 2078-12-18 ligt niet in het verleden voor Jozef Stalin"`

3.3. Maak een nieuwe klasse `Demo_4` met een main. Maak daar een aantal foutieve basisobjecten aan om de logging te testen.

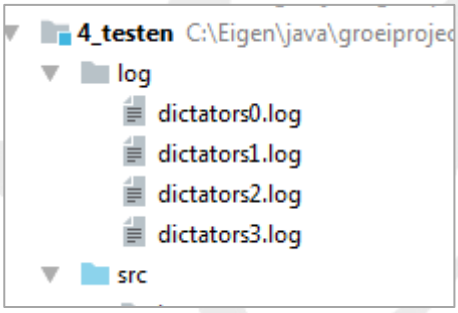
Voorbeeld resultaat (logging en gewone output kunnen gemengd op de console getoond worden):

```
aug 25, 2017 8:56:20 PM be.kdg.dictators.model.Dictator setNaam
SEVERE: Naam is verplicht

aug 25, 2017 8:56:20 PM be.kdg.dictators.model.Dictator setGeslacht
Dictators gesorteerd op naam:
SEVERE: Geslacht is verplicht voor Nicolae Ceausescu
Onbekend              (°1902) Iran            regime: Fundamentalisme        8,5 mln doden
aug 25, 2017 8:56:20 PM be.kdg.dictators.model.Dictator setLand
SEVERE: Land is verplicht voor Nicolae Ceausescu
Adolf Hitler          (°1889) Duitsland       regime: Nazisme               66,0 mln doden
aug 25, 2017 8:56:20 PM be.kdg.dictators.model.Dictator setGeboorte
Augusto Pinochet      (°1915) Chili           regime: Militaire dictatuur    0,1 mln doden
SEVERE: Geboortedatum 2078-12-18 ligt niet in het verleden voor Jozef Stalin.
```

3.4. Voeg logging van niveau FINER toe die registreert welke objecten toegevoegd worden aan of verwijderd uit de multiklasse.

3.5. Maak naast de **src** map een **log** map. Zorg ervoor dat alle boodschappen van niveau FINER of hoger naar een log map in de projectdirectory geschreven worden. Een log bestand mag maximum 10KB groot zijn en je houdt maximaal 4 bestanden bij. Voer `Demo_4` meerdere keren uit en bekijk de logfiles →



Voorbeeld van een logfile:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2017-08-28T19:32:28</date>
  <millis>1503941548472</millis>
  <sequence>0</sequence>
  <logger>be.kdg.dictatorproject.model.Dictator</logger>
  <level>SEVERE</level>
  <class>be.kdg.dictatorproject.model.Dictator</class>
  <method>setNaam</method>
  <thread>1</thread>
  <message>Naam is verplicht</message>
</record>
<record>
```

3.6. Maak een nieuwe package `logging` en schrijf daarin een klasse `SmallLogFormatter`, die `java.util.logging.Formatter` extend en die op 1 lijn logt in volgend formaat:

```
date time LEVEL> boodschap
```

Voorbeeld:
`2017-08-25 23:09:35.645 SEVERE> Naam is verplicht.`

- Je doet een override van de abstracte methode: `public String format(LogRecord record)` `LogRecord` bevat methoden om alle info uit de log record op te halen.
- Om uit de record een `LocalDateTime` te halen gebruik je:
  `Instant.ofEpochMilli(record.getMillis()).atZone(ZoneId.systemDefault()).toLocalDateTime()`
- Om uit de record de log boodschap te halen gebruik je `java.text.MessageFormat`:
  `MessageFormat.format(record.getMessage(), record.getParameters())`
- Stel een String samen met de boodschap in het gewenste formaat, sluit af met een newline en return de String.
- Configureer de ConsoleHandler in `logging.properties`, zodat deze je formatter gebruikt.

Voorbeeld van gewenste console logging:

```
2017-08-25 23:09:35.645 SEVERE> Naam is verplicht
2017-08-25 23:09:35.714 SEVERE> Geslacht is verplicht voor Nicolae Ceausescu
2017-08-25 23:09:35.730 SEVERE> Land is verplicht voor Nicolae Ceausescu
2017-08-25 23:09:35.730 SEVERE> Geboortedatum 2078-12-18 ligt niet in het verleden
voor Jozef Stalin.
2017-08-25 23:09:35.730 SEVERE> Duur 0 moet groter zijn dan 0 voor Adolf Hitler.
```