GROEIPROJECT
MODULE 9: MULTITHREADING

1. Voorbereiding

1.1. Open het groeiproject en maak daarin een negende module: `9_threads`

1.2. Kopieer van module `1_herhaling` de hele `src` map (met o.a. de oorspronkelijke basis- en multiklasse). Kopieer van module `5_patterns` enkel de klasse `XxxFactory` (in ons voorbeeld `DictatorFactory`) en zet ze in de package `model`.

2. Threads

De bedoeling is dat we **multithreading** gaan toepassen: Verschillende threads gebruiken de `XxxFactory` en produceren elk een List van 1000 random basisobjecten die elk aan een specifieke voorwaarde voldoen.
In ons voorbeeld is er een thread die 1000 random *vrouwelijke* dictators produceert, een andere maakt 1000 random *jonge* dictators, en een andere maakt 1000 random dictators waarvan de *naam met een "A" begint*.

2.1. Maak een nieuwe package `threading` en daarin een klasse `XxxRunnable` (in ons voorbeeld: `DictatorRunnable`). In deze klasse genereren we de List met 1000 basisobjecten.

a) Implementeer de `Runnable` interface
b) Via de constructor komt een `Predicate` als parameter binnen; hierop zal gefilterd worden (bijvoorbeeld: *enkel vrouwen*)
c) In de methode `run` genereer je de random List van exact 1000 basisobjecten die voldoen aan het `Predicate`. Gebruik hiervoor de `XxxFactory` (in ons voorbeeld: `DictatorFactory.newRandomDictator`)
TIP: `Stream.generate`
d) Voorzie ook een getter om de gecreëerde List op te vragen.

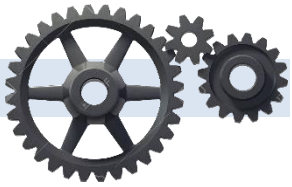
2.2. Maak een nieuwe klasse `Demo_9` met een main om uit te testen:

a) Maak een drietal threads met de `XxxRunnable` klasse; elk krijgt een ander `Predicate` mee en zal dus een specifieke List aanmaken.
b) Start de threads op en laat de main-thread wachten tot ze allemaal ten einde zijn.
c) Meet de tijd die de threads nodig hadden om hun job te doen (`System.currentTimeMillis`)
d) Vraag de aangemaakte Lists op en druk daarvan de eerste 5 objecten ter controle af. (Gebruik streaming; dus geen lus!)
e) Doe dit alles in een lus die 100 keer draait (voorzie hiervoor een globale variabele `TESTCOUNT`)
f) Toon tenslotte de gemiddelde tijd in millisec.

Mogelijke output:

```
3 threads verzamelen elk 1000 dictators (gemiddelde uit 100 runs): 107,6ms
```

3. Synchronization



3.1. Maak de basisklasse **immutable** (dus in ons geval: de klasse `Dictator`):

a) Verhinder overerving.
b) Zorg ervoor dat alle attributen ingekapseld zijn en maak ze onwijzigbaar nadat ze geïnitialiseerd worden via de constructor.
c) Verwijder dus ook alle setters.
d) Alle mutable objecten die als parameter binnenkomen via de constructor moet je opnieuw aanmaken (primitieve datatypes niet, `String` en `LocalDate` ook niet want die zijn al immutable)
e) De mutable objecten van vorige stap moet je in de bijhorende getter ook opnieuw aanmaken en als kopie retourneren.

3.2. We willen nu experimenteren met het **synchroniseren** van threads. Maak in de package `threading` een nieuwe klasse `XxxAttacker` (in ons geval `DictatorAttacker`). In deze klasse gaan we een List met basisobjecten "aanvallen" en bepaalde elementen verwijderen.

a) Implementeer de `Runnable` interface
b) Via de constructor komen 2 parameters binnen: de geviseerde List, en een `Predicate` dat zal gebruikt worden bij de aanval (bijvoorbeeld: *verwijder alle vrouwen*)
c) In de methode `run` doorloop je de List van basisobjecten en verwijder je degene die voldoen aan het `Predicate`. TIP: `removeIf`

3.3. Maak een nieuwe klasse `Demo_10` met een main om uit te testen:

a) Gebruik de `XxxFactory` om een List met 1000 random basisobjecten aan te maken.
b) Maak een drietal `XxxAttackers` aan die telkens een andere aanval uitvoeren op deze List.
c) Wacht tot alle threads hun job hebben gedaan en controleer dan via telling of de uitzuivering effectief gelukt is.

In de output zal je merken dat het **NIET** gelukt is.
Waarschijnlijk krijg je bovendien een `ConcurrentModificationException` aan je broek:

```
Exception in thread "JongeDictatorKiller" java.util.ConcurrentModificationException
        at java.util.ArrayList.removeIf(ArrayList.java:1411)
        at ...

Na uitzuivering:
Aantal vrouwen: 2
Aantal jonge dictators (>1990): 58
Aantal namen met "A": 0
```

3.4. Los het probleem op door **synchronization** toe te passen.

Verwachte output:

```
Na uitzuivering:
Aantal vrouwen: 0
Aantal jonge dictators (>1990): 0
Aantal namen met "A": 0
```

4. Concurrency

4.1. We willen werken met **`Callable`** zodat we kunnen gebruik maken van ThreadPools. Maak in de package `threading` een nieuwe klasse `XxxCallable` (in ons geval `DictatorCallable`):

a) Laat deze klasse de interface `Callable` implementeren

b) Voor de rest: dezelfde werking als `XxRunnable` (zie 2.1), dus ook een `Predicate` dat via de constructor binnenkomt.

c) In de methode `call` genereer je weer een random List van exact 1000 basisobjecten die voldoen aan het `Predicate`, gebruik makend van de `XxxFactory`. Retourneer de List.

4.2. Maak tenslotte een nieuwe klasse `Demo_11`:

a) Maak een drietal `XxxCallable` objecten; elk krijgt een ander `Predicate` mee en zal dus een specifieke List aanmaken.

b) Maak gebruik van een **fixed threadpool** om alle `XxxCallable` threads parallel te laten uitvoeren. Vang de resultaten op in een `Future` object.

c) Meet de tijd die de threads nodig hadden om hun job te doen (`System.currentTimeMillis`)

d) Doe dit alles in een lus die 100 keer draait (voorzie hiervoor een globale variabele `TESTCOUNT`)

e) Toon tenslotte de gemiddelde tijd in millisec.

Mogelijke output:

```
3 Futures verzamelen elk 1000 dictators (gemiddelde uit 100 runs): 106,0ms
```