GROEIPROJECT
MODULE 6: LAMBDA'S & STREAMS

1. Voorbereiding

1.1. Open het groeiproject en maak daarin een zesde module: `6_streams`

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

2. Lambda expressions en method references

2.1. In Module 1 van het groeiproject schreef je verschillende sorteermethoden in de multiklasse. Schrijf nu één algemene sorteermethode, die als parameter een `Function` krijgt, die het sorteercriterium uit je basisklasse haalt. Te implementeren methode in het voorbeeld `Dictators`:

```
public List<Dictator> gesorteerdOp(Function<Dictator, Comparable> f)
```

2.2. Maak een nieuwe main klasse `Demo_6` en test de nieuwe methode een aantal keer uit.
Geef de Function door als een method reference.

2.3. Maak een nieuwe package `util` met daarin een klasse `Functies` met daarin een aantal generieke static methoden.
Van de eerste methode `filteredList` ziet de header er als volgt uit:

```
public static <T> List<T> filteredList(List<T> dictatorList, Predicate<T> predicate)
```

a. Merk op hoe gebruik wordt gemaakt van generics.
b. Als eerste parameter komt een List van basisobjecten binnen (in ons voorbeeld: `dictatorList`)
c. Als tweede parameter een `Predicate`-functie die zal bepalen op welke conditie gefilterd zal worden
d. Maak voorlopig gebruik van een for-loop
e. De returnwaarde is een gefilterde List van basisobjecten.

2.4. In de klasse `Demo_6`:
a. Vraag aan de klasse `Data` een gevulde List met basisobjecten.
b. Roep een aantal keer de zopas geschreven filtermethode op en geef telkens een ander `Predicate` mee.
c. Geef het `Predicate` bij de oproep mee als een Lambda-expression.

Voorbeeld van oproep:

```
dictatorList = Functions.dictatorFilter(dictatorList,
                    dictator -> dictator.getSlachtoffers() > 3);
```

Mogelijke afdruk:

```
Toepassing 3 keer dictatorFilter met Predicate:
Alle mannelijke dictators met een stalinistisch regime die meer dan 3 mln slachtoffers
maakten:
Jozef Stalin          (°1878) Rusland        regime: Stalinisme        45,0 mln doden
Kim Il Sung           (°1912) Noord-Korea    regime: Stalinisme         4,0 mln doden
```

2.5. Maak een tweede methode `averageCollection` met de volgende signature:

```
public static <T> Double averageCollection (Collection<T> dictatorList,
ToDoubleFunction<T> mapper)
```

a. Als eerste parameter komt een `Collection`van basisobjecten binnen (in ons voorbeeld: `dictatorList`)
b. Als tweede parameter een `Function` die zal bepalen op welk aspect het gemiddelde zal berekend worden
c. Maak voorlopig gebruik van een for-loop
d. De returnwaarde is het gemiddelde.

2.6. In de klasse `Demo_6` ga je deze methode uittesten:

a. Vraag aan de klasse `Data` opnieuw een vers gevulde List met basisobjecten.
b. Roep een aantal keer de zopas geschreven methode op en geef telkens een andere `Function` mee, die bepaalt van welk attribuut het gemiddelde moet berekend worden.
c. Geef de `Function` bij de oproep mee als een Lambda-expression.

Voorbeeld van oproep:

```
System.out.printf("Gemiddeld aantal slachtoffers: %.1f mln\n",
       Functies.averageCollection (dictatorList, Dictator::getSlachtoffers));
```

2.7. Maak een derde methode `countIf` met de volgende signature:

```
public static <T> long countIf(Collection<T> dictatorList, Predicate<T> predicate)
```

a. Als eerste parameter komt een `Collection` van basisobjecten binnen (in ons voorbeeld: `dictatorList`)
b. Als tweede parameter een `Predicate` dat bepaalt op welk aspect geselecteerd wordt
c. Maak voorlopig gebruik van een for-loop
d. De returnwaarde is het aantal objecten dat aan de selectie voldoet.

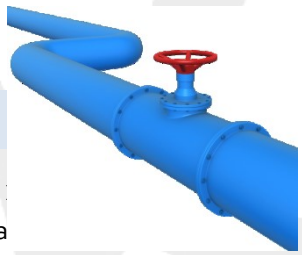
2.8. In de klasse `Demo_6` ga je deze methode uittesten:

a. Roep een aantal keer de zopas geschreven methode op en geef telkens een ander `Predicate` mee.
b. Geef het `Predicate` bij de oproep mee als een Lambda-expression.

Voorbeeld van oproep:

```
System.out.printf("Aantal dicators met wreedheid > 50%%: %d\n",
        Functies.countIf(dictatorList, dictator -> dictator.getWreedheid() > 50.0));
```

3. Streams



In de klasse `Demo_6` vraag je opnieuw een verse List van basisobjecten op aan de klasse
Je gaat nu via streaming de onderstaande opdrachten uitvoeren, toegepast op jouw data
Programmeer alles zo compact en efficiënt mogelijk!

3.1. Tel hoeveel elementen aan een bepaald criterium beantwoorden
Voorbeeld van afdruk:

```
Aantal dictators dat na 1900 geboren is: 9
```

3.2. Sorteer (op 2 attributen!) en druk af
Voorbeeld van afdruk:

```
Alle dictators gesorteerd op regime en dan op naam:
Mao Tse-tung           (°1893) China          regime: Communisme              8,0 mln doden
```

```
Nicolae Ceausescu      (°1918) Roemenië       regime: Despotisme            0,5 mln doden
Laurent-Désiré Kabila  (°1939) Congo          regime: Dictatuur             0,3 mln doden
Benito Mussolini       (°1883) Italië         regime: Fascisme              2,0 mln doden
Ayatollah Khomeini     (°1902) Iran           regime: Fundamentalisme       8,5 mln doden
enz…
```

3.3. Zet een attribuut om naar hoofdletters, verwijder de dubbels, sorteer in aflopende volgorde en zet de elementen in een string, gescheiden door komma's

Voorbeeld van afdruk:

```
Alle dictatorlanden in hoofdletters, omgekeerd gesorteerd en zonder dubbels:
SPANJE, RUSLAND, ROEMENIË, OEGANDA, NOORD-KOREA, ITALIË, IRAN, FILIPIJNEN, DUITSLAND,
CONGO, CHINA, CHILI, CAMBODJA
```

3.4. **`filter`** en **`findAny`**

Voorbeeld van afdruk:

```
Een willekeurige dictator met meer dan 75% wreedheid:
Mao Tse-tung : 80.0
```

3.5. Toon de naam van het element met de hoogste waarde voor een attribuut

Voorbeeld van afdruk:

```
De dictator met de meeste slachtoffers: Adolf Hitler
```

3.6. Selecteer volgens een criterium, neem de naam, sorteer en geef terug als List.

Voorbeeld van afdruk:

```
List met gesorteerde dictatornamen die beginnen met 'A':
[Adolf Hitler, Augusto Pinochet, Ayatollah Khomeini]
```

3.7. Sorteer op een attribuut en partitioneer in twee delen **:**

Voorbeeld van afdruk:

```
Sublist met dictators geboren VOOR 1900:
Jozef Stalin           (°1878) Rusland        regime: Stalinisme           45,0 mln doden
Benito Mussolini       (°1883) Italië         regime: Fascisme              2,0 mln doden
Adolf Hitler           (°1889) Duitsland      regime: Nazisme              66,0 mln doden
Francisco Franco       (°1892) Spanje         regime: Militaire dictatuur   5,0 mln doden
Mao Tse-tung           (°1893) China          regime: Communisme            8,0 mln doden

Sublist met dictators geboren NA 1900:
Ayatollah Khomeini     (°1902) Iran           regime: Fundamentalisme       8,5 mln doden
Kim Il Sung            (°1912) Noord-Korea    regime: Stalinisme            4,0 mln doden
Augusto Pinochet       (°1915) Chili          regime: Militaire dictatuur   0,1 mln doden
Ferdinand Marcos       (°1917) Filipijnen     regime: Kleptocratie          0,1 mln doden
Nicolae Ceausescu      (°1918) Roemenië       regime: Despotisme            0,5 mln doden
Idi Amin               (°1925) Oeganda        regime: Kannibalisme          3,0 mln doden
Pol Pot                (°1925) Cambodja       regime: Militaire dictatuur   4,0 mln doden
Mobutu Sese-Seko       (°1930) Congo          regime: Kleptocratie          0,5 mln doden
Laurent-Désiré Kabila  (°1939) Congo          regime: Dictatuur             0,3 mln doden
```

4. Streams en lambda's

Tenslotte gaan we een aantal eerder geschreven methoden optimaliseren.

4.1. Refactor jouw code in `Demo_6`. Vervang OVERAL de lussen door gebruik te maken van streaming in combinatie met lambda's en/of method references. Er mag dus geen enkele lus meer staan!

4.2. Herschrijf in de klasse `Functies` alle methoden. Maak NERGENS nog gebruik van een lus; gebruik overal streaming! Codeer zo compact en efficiënt mogelijk.

Run `Demo_6` opnieuw en controleer de werking.