Opgave 8 Java 8

Omdat we de file SightingsNew moesten inlezen moesten er een paar aanpassingen gebeuren aan het programma, omdat we voor de plaats nu een String inlezen i.p.v. een integer. Nadat we dit hadden aangepast gingen we kijken of we alles konden afdrukken.

```
public class Main {

    public static void main(String[] args) {

        AnimalMonitor aM = new AnimalMonitor();
        aM.addSightings( filename: "sightingsNew.csv");
        System.out.println("*** all observations ");
        aM.printList();
```

Dit geeft een hele hoop informatie terug

```
*** all observations
Mountain Gorilla, count = 3, area = Mangroves National Park, spotter = Jean, period = 0
Buffalo, count = 10, area = Mangroves National Park, spotter = Jean, period = 0
Elephant, count = 0, area = Mangroves National Park, spotter = Jean, period = 0
Mountain Gorilla, count = 1, area = Odzala National Park, spotter = Karel, period = 0
Mountain Gorilla, count = 3, area = Garamba National Park, spotter = Piet, period = 0
Mountain Gorilla, count = 0, area = Odzala National Park, spotter = Francis, period = 0
Buffalo, count = 2, area = Mangroves National Park, spotter = Francis, period = 0
Topi, count = 25, area = Mangroves National Park, spotter = Francis, period = 0
Mountain Gorilla, count = 4, area = Mangroves National Park, spotter = Jean, period = 1
Buffalo, count = 16, area = Mangroves National Park, spotter = Jean, period = 1
Topi, count = 20, area = Mangroves National Park, spotter = Karel, period = 1
Buffalo, count = 0, area = Odzala National Park, spotter = Francis, period = 1
Topi, count = 30, area = Odzala National Park, spotter = Francis, period = 1
Mountain Gorilla, count = 1, area = Mangroves National Park, spotter = Jan, period = 2
Mountain Gorilla, count = 2, area = Odzala National Park, spotter = Karel, period = 2
Mountain Gorilla, count = 0, area = Garamba National Park, spotter = Piet, period = 2
Topi, count = 30, area = Odzala National Park, spotter = Francis, period = 2
Elephant, count = 24, area = Odzala National Park, spotter = Francis, period = 2
```

Filtering

We hebben al een filter methode gekregen genaamd printSightingsOf, hiermee kunnen we een dier als parameter meegeven en krijgen we alle observaties van dat dier terug.

```
System.out.println("\n*** all Buffalo observations ");
aM.printSightingsOf( animal: "Buffalo");

System.out.println("\n*** all Topi observations ");
aM.printSightingsOf( animal: "Topi");
```

```
*** all Buffalo observations
Buffalo, count = 10, area = Mangroves National Park, spotter = Jean, period = 0
Buffalo, count = 2, area = Mangroves National Park, spotter = Francis, period = 0
Buffalo, count = 16, area = Mangroves National Park, spotter = Jean, period = 1
Buffalo, count = 0, area = Odzala National Park, spotter = Francis, period = 1

*** all Topi observations
Topi, count = 25, area = Mangroves National Park, spotter = Francis, period = 0
Topi, count = 20, area = Mangroves National Park, spotter = Karel, period = 1
Topi, count = 30, area = Odzala National Park, spotter = Francis, period = 1
Topi, count = 30, area = Odzala National Park, spotter = Francis, period = 2
```

Op deze methode maken we ook verschillende variaties. Zo hebben we één waarop we filteren op de plaats waar ze zijn gespot (printSightingsAt) en een andere die filter op de persoon die het dier heeft gespot. De methodes zien er als volgt uit:

```
public void printSightingsOf(String animal)
{
    sightings.stream()
            .filter(sighting -> animal.equals(sighting.getAnimal()))
            .forEach(sighting -> System.out.println(sighting.getDetails()));
}

public void printSightingsAt(String area)
{
    sightings.stream()
            .filter(sighting -> area.equals(sighting.getArea()))
            .forEach(sighting -> System.out.println(sighting.getDetails()));
}

public void printSightingsBy(String spotter)
{
    sightings.stream()
            .filter(sighting -> spotter.equals(sighting.getSpotter()))
            .forEach(sighting -> System.out.println(sighting.getDetails()));
}
```

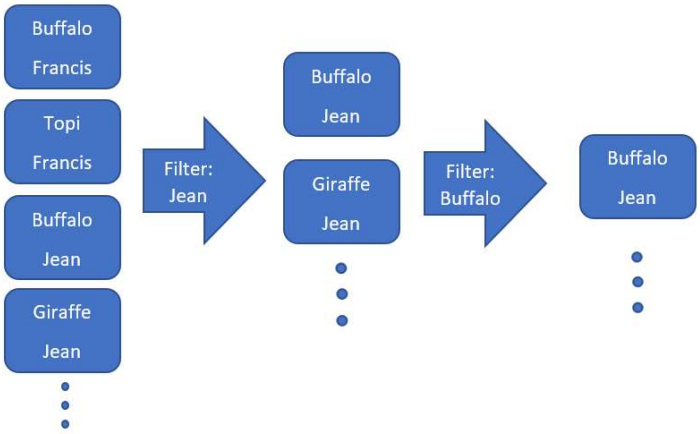
We zien dat de methode bestaat uit een filter die gaat kijken of de string van de meegegven parameter gelijk is als die van de gegevens in de stream. Na het filteren worden deze met een for each allemaal afgedrukt. De info die wordt afgedrukt krijgen we uit de method getDetails, die alle onfo achter elkaar zet.

We testen deze methodes uit in de main.

```
System.out.println("\n*** all observations at Odzala National Park");
aM.printSightingsAt( area: "Odzala National Park");

System.out.println("\n*** all observations by Francis");
aM.printSightingsBy( spotter: "Francis");
```

We kunnen ook de gefilterde stream nog eens filteren. In dit voorbeeld filteren we eerst op de spotter Jean en daarna op het dier Buffalo. Visueel ziet dit er zo uit.



De methode is niet veel verschillend t.o.v. de vorige, hier komt er enkel een extra filter achter de vorige.

```
public void printAllAnimalsBy(String spotter, String animal)
{
    sightings.stream()
            .filter(sighting -> spotter.equals(sighting.getSpotter()))
            .filter(sighting -> animal.equals(sighting.getAnimal()))
            .forEach(sighting -> System.out.println(sighting.getDetails()));
}
```

Als we een voorbeeld van hierboven uittesten krijgen we:

```
*** all buffalos by Jean
Buffalo, count = 10, area = Mangroves National Park, spotter = Jean, period = 0
Buffalo, count = 16, area = Mangroves National Park, spotter = Jean, period = 1
```

Mapping

De opdracht hier is het maken van de methode shortDetails. Dit geeft in plaats van alle info enkel de plaats terug.

```
public String shortDetails()
{
    return animal +
            ", area = " + area ;
}
```

We kunnen nu de vorige methode printSightingBy aanpassen zodat het enkel de area terugkrijgen i.p.v. alle info.

```
public void printShortSightingsBy(String spotter)
{
    sightings.stream()
            .filter(sighting -> spotter.equals(sighting.getSpotter()))
            .forEach(sighting -> System.out.println(sighting.shortDetails()));
}
```

Hieronder kunnen we het verschil zien tussen de 2 methodes. Als we bijvoorbeeld enkel de area willen weten is dit wel overzichtelijker.

```
*** all observations (short) by Francis
Mountain Gorilla, area = Odzala National Park
Buffalo, area = Mangroves National Park
Topi, area = Mangroves National Park
Buffalo, area = Odzala National Park
Topi, area = Odzala National Park
Topi, area = Odzala National Park
Elephant, area = Odzala National Park

*** all observations by Francis
Mountain Gorilla, count = 0, area = Odzala National Park, spotter = Francis, period = 0
Buffalo, count = 2, area = Mangroves National Park, spotter = Francis, period = 0
Topi, count = 25, area = Mangroves National Park, spotter = Francis, period = 0
Buffalo, count = 0, area = Odzala National Park, spotter = Francis, period = 1
Topi, count = 30, area = Odzala National Park, spotter = Francis, period = 1
Topi, count = 30, area = Odzala National Park, spotter = Francis, period = 2
Elephant, count = 24, area = Odzala National Park, spotter = Francis, period = 2
```

Reducing

De eerste methode die we hier ontwerpen is om te tellen hoe vaak een bepaald dier is gespot.

```
public int getCountOfAnimal(String animal)
{
    return sightings.stream()
            .filter(sighting -> animal.equals(sighting.getAnimal()))
            .map(sighting -> sighting.getCount())
            .reduce( t: 0, (runningSum, count) -> runningSum + count);
}
```

We tellen gewoon alle aantallen op, maar doen eerst een filter op dieren zodat we enkel het meegegeven dier tellen.

We kunnen ook zorgen dat een bepaald dier enkel wordt geteld als het gespot is door een bepaald persoon. Dit is zoals het vorige maar dan met een extra filter.

```
public int getCountOfAnimalsSeenBy(String animal, String spotter)
{
    return sightings.stream()
                .filter(sighting -> animal.equals(sighting.getAnimal()))
                .filter(sighting -> spotter.equals(sighting.getSpotter()))
                .map(sighting -> sighting.getCount())
                .reduce( t: 0, (runningSum, count) -> runningSum + count);
}
```

In de main ziet dit er zo uit:

```
System.out.println("\n*** amount of buffalo's spotted");
System.out.println(aM.getCountOfAnimal("Buffalo"));

System.out.println("\n*** amount of Topi's spotted by Francis");
System.out.println(aM.getCountOfAnimalsSeenBy( animal: "Topi", spotter: "Francis"));
```

Met als resultaten:

```
*** amount of buffalo's spotted
28

*** amount of Topi's spotted by Francis
85
```

De volgende opdracht was een ArrayList maken met de gegevens in van een bepaalde spotter.

```
public List<Sighting> listSightingsBy (String spotter)
{

    return sightings.stream()
            .filter(sighting -> spotter.equals(sighting.getSpotter()))
            .collect(Collectors.toCollection(ArrayList::new));

}
```

De laaste opdracht was de gegevens van een bepaalde spotter in een Map zetten, de methode ziet er zo uit:

```
public void makeMapOf (String spotter)
{
    List<Sighting> spotting = listSightingsBy(spotter);
    Map<String, Integer> mapInfo =
            (Map<String, Integer>) spotting.stream()
                    .collect(Collectors.groupingBy
                            (Sighting::getAnimal,Collectors.summingInt
                                    (Sighting::getCount)));

    System.out.println(spotter+":");
    for(Map.Entry<String, Integer> entry : mapInfo.entrySet())
    {
        System.out.println(entry.getKey()+"   "+entry.getValue());
    }
}
```

In het bovenste deel wordt er een map opgemaakt die het dier als Key opslaat en een aantal keren gespot als Value. De aantal worden ook opgeteld bij meerdere spottingen. In het tweede gedeelde wordt elke entry van die map afgedrukt.

In de main geeft dit een mooi overzicht:

```
*** map of animals spotted by Jean
Jean:
Mountain Gorilla   7
Elephant   0
Buffalo   26
```