Lab Report Kubilay Günther/ Pavel Tsvyatkov - 21.05.19

1. Semester / IMI - WH C 579

Professor	Name of Exercise
Prof. Dr. Debora Weber-Wulff	Lab 5 - Lambda Functions

Lab-Plan

1. Assignment

1.1. Introduction

1.2. Download the **animal-monitoring-v1** project and check that it runs. Now rewrite the `printList` method in the `AnimalMonitor` class to use a lambda, just as we did in the lecture. Now apply each of the variations that you recorded in P2, compile them and run them. Did they all work? Record the results for your lab report.

1.3. Rewrite the `printSightingsOf` method in the `AnimalMonitor` class to use streams and lambdas. Test to make sure that your project still works as before.

1.4. Write a method in the `AnimalMonitor` class to print the details of all the sightings recorded on a particular `dayID`, which is passed as a parameter to the method.

1.5. Write a method that used two `filter` calls to print details of all the sightings of a particular animal made on a particular day—the method takes the animal name and the day ID as parameters. Does the order of the two `filter` calls matter in your solution? Justify your answer.

1.6. Write a method to print the counts of all sightings of a particular animal. Your method should use the `map` operations as part of the pipeline. If a pipeline contains a `filter` operation and a `map` operation, does the order of the operations matter to the final result? Justify your answer.

1.7. Rewrite the `printEndangered` method in your project to use streams and test that it works correctly. Detail how you tested this method in your report.

1.8. (For the bored) There is a special "`::`" notation in Java that can be used with lambdas. Research the syntax, and rewrite `printSightingsBy`to use this syntax. Does the operation of the method change?

1.9. (For the bored) Add a method to `AnimalMonitor` that takes two parameters, a spotter-ID and a day-ID and returns a `String` containing the names of the animals seen by the spotter on that particular day. You should include only animals whose sighting count is greater than zero. Now add a method that takes an animal and a day-ID and returns the spotters who saw this animal, if any, on that particular day.

2. Reflection & Summary of the Lab

2.1. Kubilay's Reflection

2.2. Pavel's Reflection

3. Appendix

3.1. Our code

3.2. Pre-Labs

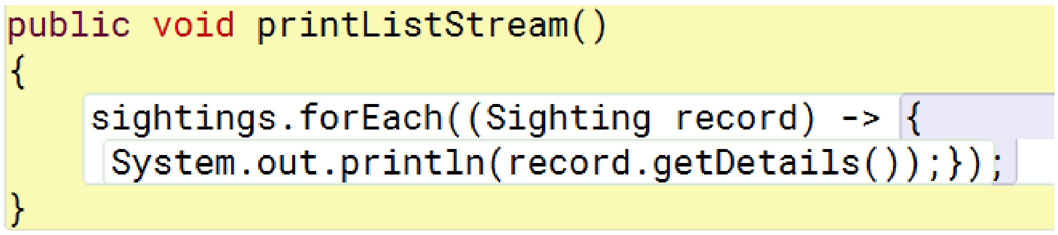
1. Assignment

1.1 Introduction

Laboratory 5 was about lambda functions. Pavel and Kubilay worked together in this lab. We worked on the assignment in the lab and there we managed to solve the tasks till exercise 5. For the last exercise we met on Thursday and found a solution to it. After that we started working on our lab report. We also collaborated with our tutor Lotte, because we needed help to test our method to print an endangered animal.

1.2 Download the animal-monitoring-v1 project and check that it runs. Now rewrite the printList method in the AnimalMonitor class to use a lambda, just as we did in the lecture. Now apply each of the variations that you recorded in P2, compile them and run them. Did they all work? Record the results for your lab report.

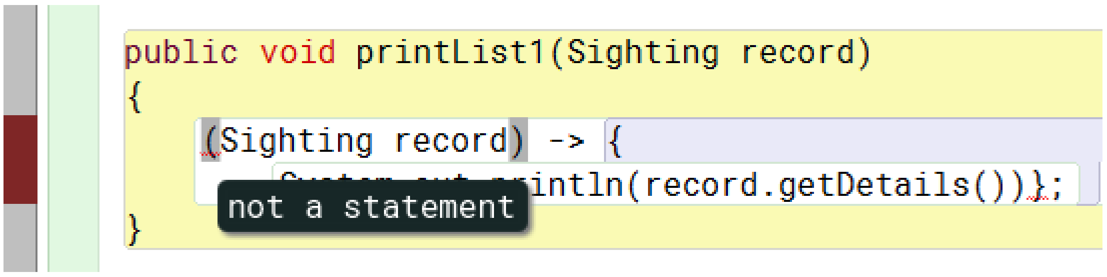
We downloaded the animal-monitoring-v1 project and opened it. We compiled it and checked that it runs and then we started with the exercise. We searched for the "printList" method in the AnimalMonitor class and started thinking how to structure our code, so that we can rewrite it with to use a lambda. We had some problems, so we had to check our notes from the lectures about lambda functions. After discussing about the task, we wrote the following code.



```
public void printListStream()
{
    sightings.forEach((Sighting record) -> {
     System.out.println(record.getDetails());});
}
```

We tested it and it was working well, so we continued with the tasks.

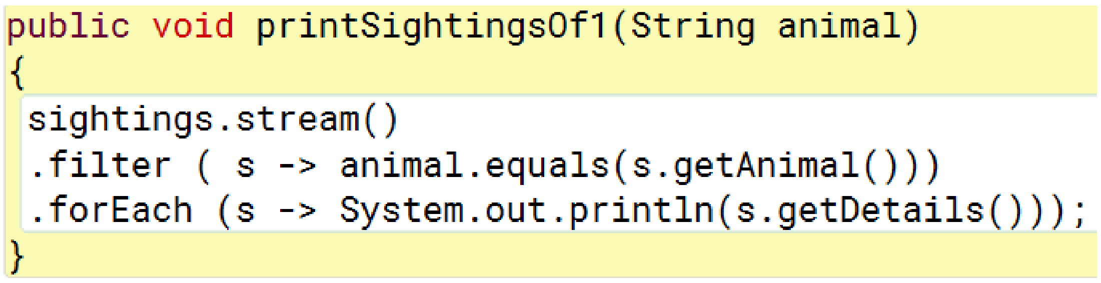
We had the same idea about P2. We only had one variation, but unfortunately it wasn't working and it was giving us an error "not a statement". We tried to work on a fix for that error, but we couldn't come up with a solution to that. We weren't entirely sure how to restructure our code to make the lambda work. We discussed and we both agreed that we should have something in front of "(Sighting record)", but after spending some time on that we didn't fix the problem.



```
public void printList1(Sighting record)
{
    (Sighting record) -> {
                        intln(record.getDetails())};
}
```

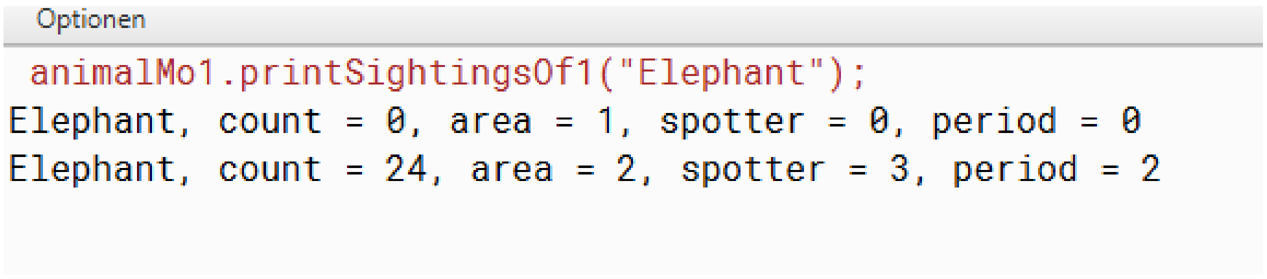
1.3 Rewrite the printSightingsOf method in the AnimalMonitor class to use streams and lambdas. Test to make sure that your project still works as before.

We searched for the printSightingsOf method in the AnimalMonitor class. Then we commented this method out to rewrite it using a lambda. Our first problem in this task was that we mistyped "animals" in the parameter. We also had a second error and that was an extra semicolon. After discussing together, we fixed our bugs we tested it and it worked as before. Our code for that exercise looked like that.



```
public void printSightingsOf1(String animal)
{
 sightings.stream()
 .filter ( s -> animal.equals(s.getAnimal()))
 .forEach (s -> System.out.println(s.getDetails()));
}
```

We tested with "Elephant" to make sure that everything works correctly and it was working well.



```
 animalMo1.printSightingsOf1("Elephant");
Elephant, count = 0, area = 1, spotter = 0, period = 0
Elephant, count = 24, area = 2, spotter = 3, period = 2
```

1.4 Write a method in the AnimalMonitor class to print the details of all the sightings recorded on a particular dayID, which is passed as a parameter to the method.

Before we started writing the method, we discussed for a moment how to structure our code. We wrote "public void printSightingsByDay" and in the parameter we passed "int dayID". We began with the "sightings.stream()" and then we both agreed that we should filter out the particular day. We were careful not to use “equals()” , since “dayID” was of type int and so was the “getPeriod()” method. We both agreed that we should use “==” for comparison. After that we had to print the details and our code was looking like this.

```
public void printSightingsByDay (int dayID)
{
    sightings.stream()
    .filter(s -> dayID == s.getPeriod())
    .forEach(s -> System.out.println(s.getDetails()));
}
```

Then we went on and tested our method for day one. We were happy, because our code was working well and we continued with the next exercise.

```
Mountain Gorilla, count = 4, area = 1, spotter = 0, period = 1
Buffalo, count = 16, area = 1, spotter = 0, period = 1
Topi, count = 20, area = 1, spotter = 1, period = 1
Buffalo, count = 0, area = 2, spotter = 3, period = 1
Topi, count = 30, area = 2, spotter = 3, period = 1
```

1.5 Write a method that used two filter calls to print details of all the sightings of a particular animal made on a particular day—the method takes the animal name and the day ID as parameters. Does the order of the two filter calls matter in your solution? Justify your answer.

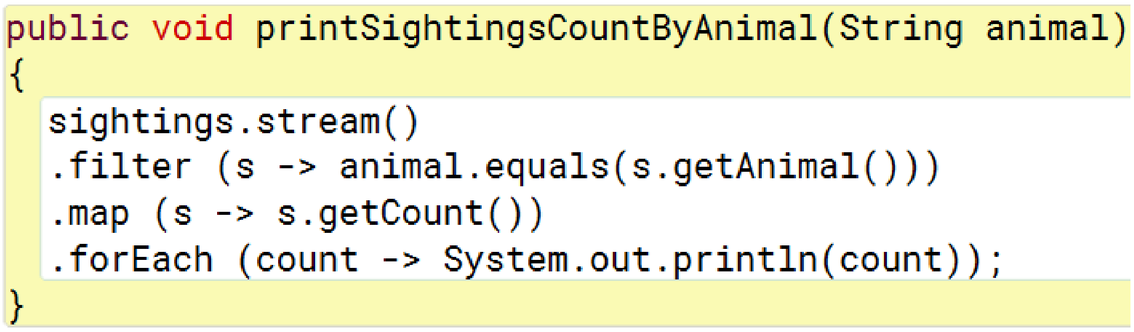
For this exercise we had to create a new method which takes two parameters - the animal name and the day ID. We also had to use two filters and then print all the sightings of a particular animal on a particular day. Then we needed to check if the order of how we filtered out matters. We created our new method “printSightingsByDayOfAnimal” and passed the two parameters “int dayID and “String animal”. We tried to filter out the dayID first and then the animal. At this point our code was looking like this.

```
public void printSightingsByDayOfAnimal(int dayID, String animal)
{
  sightings.stream()
  .filter (s-> dayID == s.getPeriod())
  .filter (s-> animal.equals(s.getAnimal()))
  .forEach (s-> System.out.println(s.getDetails()));

}
```

We tested our code and it was working well. After that we tried to switch the order of how we filter and filtered out the animal first and then the dayID. We compiled and tested our code and the result remained the same. We both discussed and agreed that the order of how we filter didn't matter.

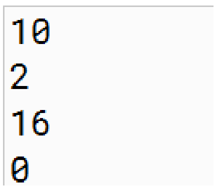
1.6 Write a method to print the counts of all sightings of a particular animal. Your method should use the map operations as part of the pipeline. If a pipeline contains a filter operation and a map operation, does the order of the operations matter to the final result? Justify your answer.

Before we started with this task we discussed about what we had to do. In this exercise we had to use filter and then also use map operations as part of the pipeline. We had to discuss and note if the order of how we filter out and map matters. We started writing our new method "printSightingsCountByAnimal" and gave it a single parameter "String animal". We decided to filter the animal first and then map the count. After that we had to print out the counts. After discussing about the structure for a bit, we wrote the following code.



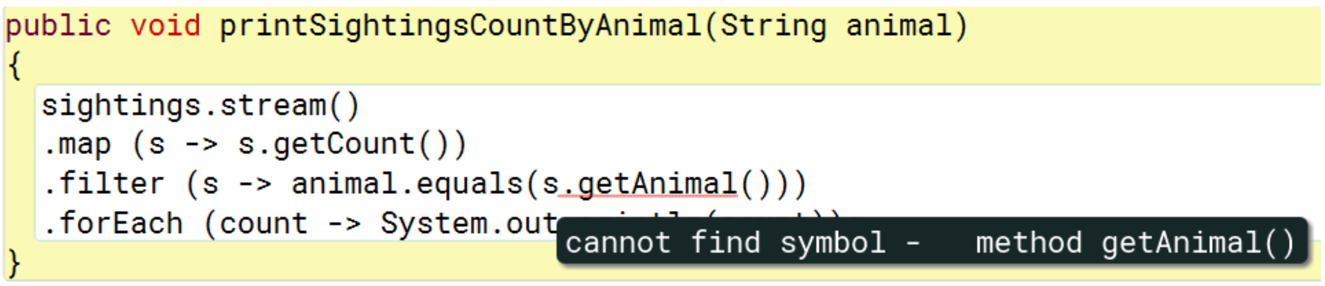
```
public void printSightingsCountByAnimal(String animal)
{
   sightings.stream()
   .filter (s -> animal.equals(s.getAnimal()))
   .map (s -> s.getCount())
   .forEach (count -> System.out.println(count));
}
```

We tested our code with the animal "Buffalo" and it was working well. As output we were seeing the counts of all sightings of the buffalo.



```
10
2
16
0
```

Then we tried to use map first and then filter, but that wasn't working and it was giving the following error.

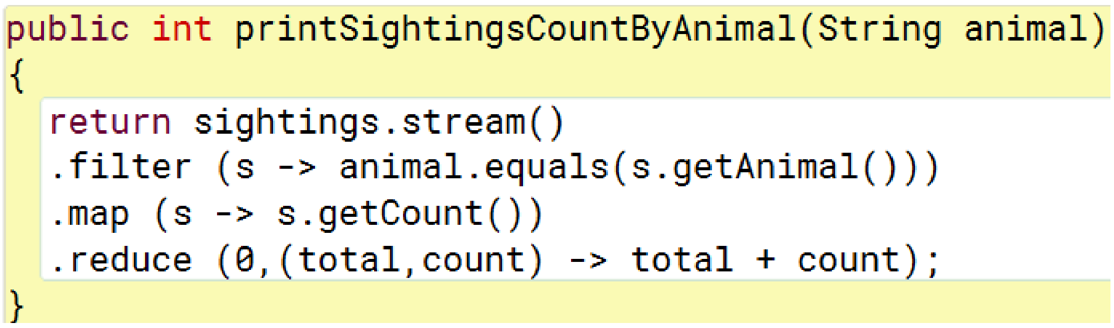


```
public void printSightingsCountByAnimal(String animal)
{
   sightings.stream()
   .map (s -> s.getCount())
   .filter (s -> animal.equals(s.getAnimal()))
   .forEach (count -> System.out
}
```

```
cannot find symbol -   method getAnimal()
```

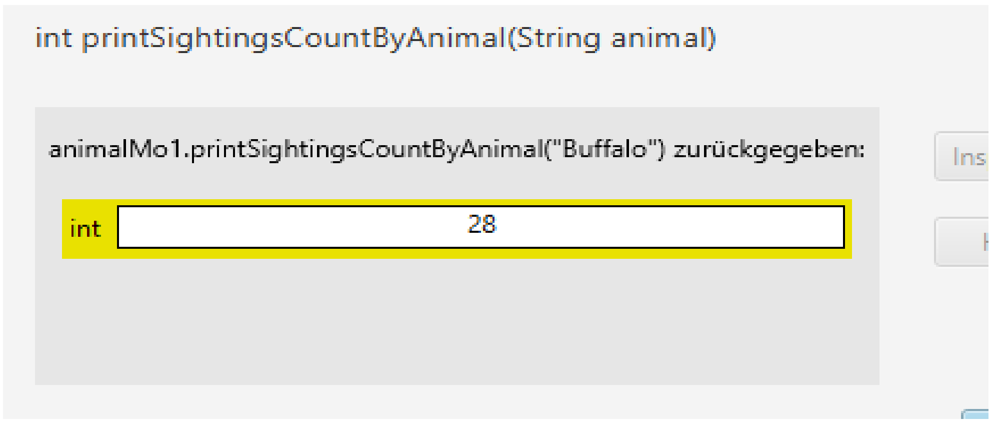
We both discussed and agreed that the order of how we map and filter matters, so we structured the code as it was before and filtered out first before using the map operation.

When printing we saw that we get the counts of all sightings of a particular animal separated. Out of curiosity we both discussed that it would be interesting to see if we can use the ".reduce" operation to get a single number as output, like we did in class. For this to work, we had to change the method to be of type int and then use "return" before "sightings.stream()". Our code was looking like this.



```java
public int printSightingsCountByAnimal(String animal)
{
   return sightings.stream()
   .filter (s -> animal.equals(s.getAnimal()))
   .map (s -> s.getCount())
   .reduce (0,(total,count) -> total + count);
}
```

We tested our code with the animal "Buffalo" and we were happy, because it was working exactly as we wanted it to work. As a result we were seeing the counts of all sightings of a particular animal reduced to one number.



```
int printSightingsCountByAnimal(String animal)

animalMo1.printSightingsCountByAnimal("Buffalo") zurückgegeben:

int 28
```

After that we changed our code back to what we did previously, but we were still glad that we could also test out how ".reduce" worked.

1.7 Rewrite the printEndangered method in your project to use streams and test that it works correctly. Detail how you tested this method in your report.

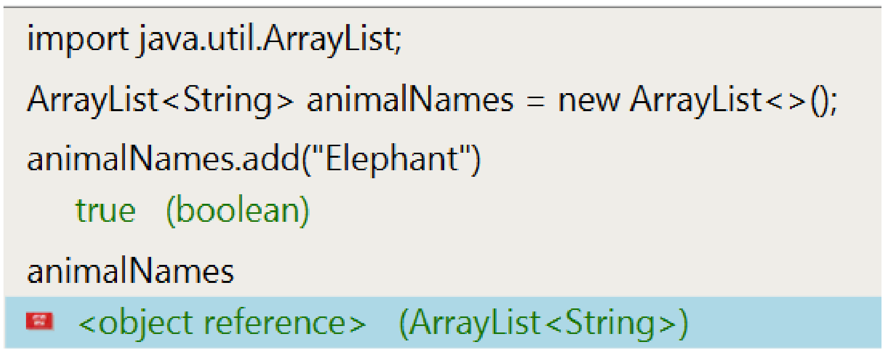
We didn't have time in the lab for this exercise, so we worked on it from home. We noticed that the method takes two parameters and one of them was a new array list "animalNames". The other parameter was "dangerThreshold" of type int. We had to discuss about this exercise and think of a way to solve it. We started writing our new method "printEndangeredStream" and used the same two parameters. We both agreed that this time we need to use "animalNames.stream()", since we are working with a different array list. We had to use filter, which checks if the count of a particular animal is less than or equal to the danger threshhold and then print out whether the animal is endangered or not.

We thought about it and wrote our code like this.

```
public void printEndangeredStream(ArrayList<String> animalNames,
                                  int dangerThreshhold)
{
 animalNames.stream()
    .filter ( animal -> getCount(animal) <= dangerThreshhold)
    .forEach ( animal -> System.out.println(animal + " is endangered."));
}
```

After we finished writing our code we wanted to test if it works correctly. We tried use our new method, but we weren't sure what value to pass for the array list.

We talked about it and both agreed to go and ask our tutor Lotte about it on Monday. She helped us to test our method and said that we could use the code pad to create the "animalNames" array list and add animals to it. After discussing about it, in the code pad we wrote the following.



```
import java.util.ArrayList;
ArrayList<String> animalNames = new ArrayList<>();
animalNames.add("Elephant")
    true   (boolean)
animalNames
    <object reference>   (ArrayList<String>)
```

We clicked on the object reference so it appears in our object bench. From there we could use it when we test our "printEndangeredStream" method. After doing this, we tested our method and it was working correctly. If the value for the danger threshhold was bigger than the counts of the particular animal it was printing out that the particular animal is endangered.

2. Reflection & Summary of the Lab

2.1. Kubilay's Reflection

The lab gave me a better understanding about lambda functions and their use. This week we had more time to solve the exercises, because we did some of the methods in the lecture. Even though we didn't need to think about a completely new method on our own, it was really nice and helpful. We tried not to look in our notes, and then when we needed help we searched for the right order or notes. At this point I want to credit Pavel. I liked to work with him due to the fact that he took the time we needed. He was not stressed about solving the tasks in the lab, maybe I didn't notice, but this fact relaxed me and helped me to focus on the code. Last week I was stressed and had to work a lot at home or with Lotte. This week I could take everything important along the lab.

2.2. Pavel's Reflection

In this exercise I worked more with lambda functions and I got a better understanding on how to use them. I had to rewrite methods, so that they use streams and lambdas, and after doing the exercises I feel more confident to use streams and lambdas. I also learned that when I only need to filter out, the order of the filters didn't matter. But when I also need to map, then the order matters and I need to know how exactly I need to structure the code. I also worked with the special "::" notation that can be used with lambdas and understood it better.

3. Appendix

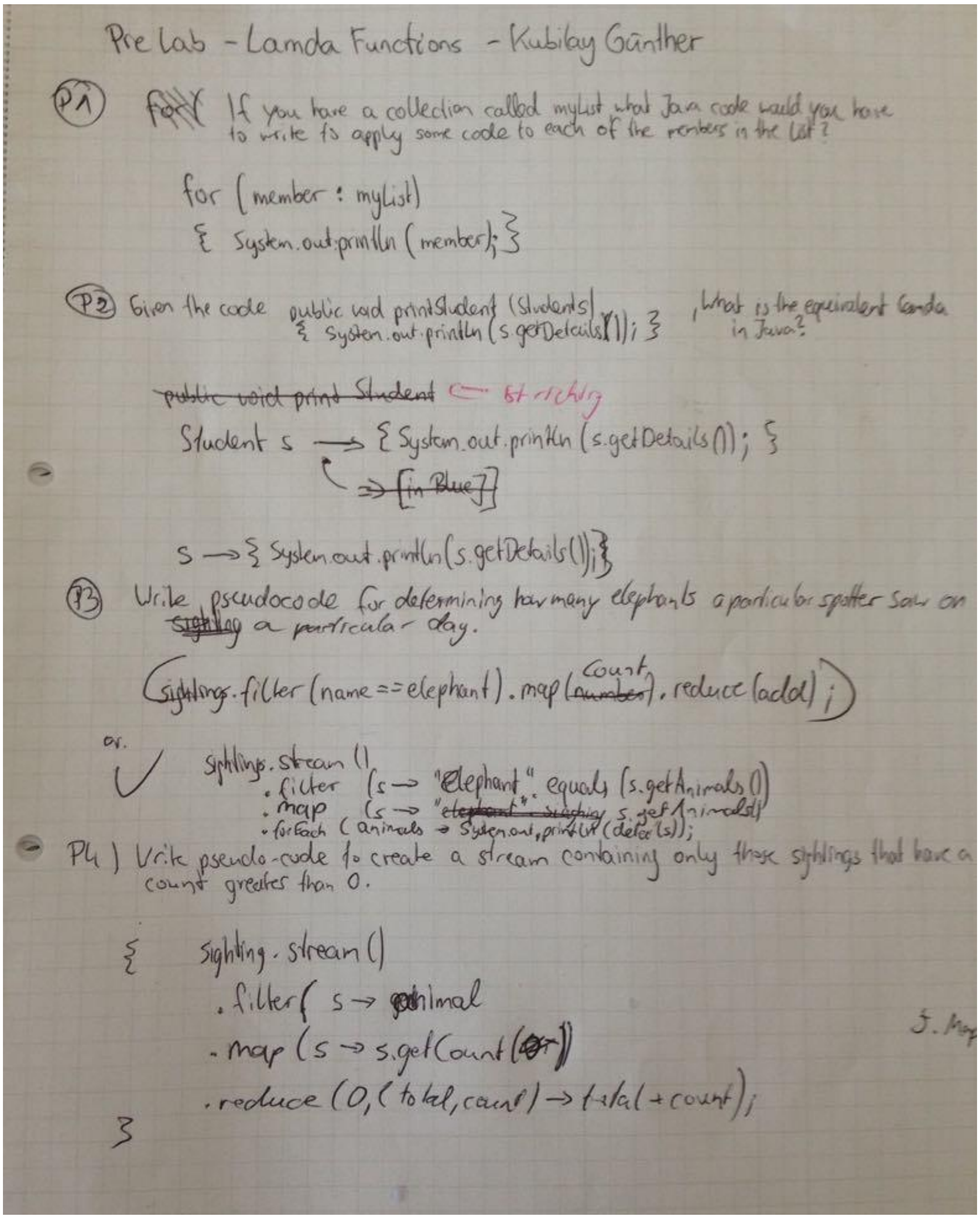
3.1. Code

```
import java.util.ArrayList;
import java.util.Iterator;

/**
 * Monitor counts of different types of animal.
 * Sightings are recorded by spotters.
 *
 * @author David J. Barnes and Michael Kölling
 * @version 2016.02.29 (imperative)
 */
public class AnimalMonitor
{
    // Records of all the sightings of animals.
    private ArrayList<Sighting> sightings;

    /**
     * Create an AnimalMonitor.
     */
    public AnimalMonitor()
    {
        this.sightings = new ArrayList<>();
    }

    /**
     * Add the sightings recorded in the given filename to the current list.
     * @param filename A CSV file of Sighting records.
     */
    public void addSightings(String filename)
    {
        SightingReader reader = new SightingReader();
        sightings.addAll(reader.getSightings(filename));
    }

    /**
     * Print details of all the sightings.
     */
    public void printList()
    {
        for(Sighting record : sightings) {
            System.out.println(record.getDetails());
        }
    }

    public void printListStream()
    {
        sightings.forEach((Sighting record) ->
        {
         System.out.println(record.getDetails());});
    }

    /**
     * Print the details of all the sightings of the given animal.
     * @param animal The type of animal.
     */
    public void printSightingsOf(String animal)
    {
        for(Sighting record : sightings) {
            if(animal.equals(record.getAnimal())) {
                System.out.println(record.getDetails());
            }
        }
    }

    public void printSightingsOf1(String animal)
    {
     sightings.stream()
     .filter ( s -> animal.equals(s.getAnimal()))
     .forEach (s -> System.out.println(s.getDetails()));
    }

    public void printSightingsByDay (int dayID)
    {
        sightings.stream()
        .filter(s -> dayID == s.getPeriod())
        .forEach(s -> System.out.println(s.getDetails()));
    }

    public void printSightingsByDayOfAnimal(int dayID, String animal)
    {
      sightings.stream()
      .filter (s-> dayID == s.getPeriod())
      .filter (s-> animal.equals(s.getAnimal()))
      .forEach (s-> System.out.println(s.getDetails()));

    }

    public void printSightingsCountByAnimal(String animal)
    {
      sightings.stream()
      .filter (s -> animal.equals(s.getAnimal()))
      .map (s -> s.getCount())
      .forEach (count -> System.out.println(count));
    }

    /**
     * Print all the sightings by the given spotter.
     * @param spotter The ID of the spotter.
     */
    public void printSightingsBy(int spotter)
    {
        for(Sighting record : sightings) {
            if(record.getSpotter() == spotter) {
                System.out.println(record.getDetails());
            }
        }
    }

    /**
     * Print a list of the types of animal considered to be endangered.
     * @param animalNames A list of animals names.
     * @param dangerThreshold Counts less-than or equal-to to this level
     *                        are considered to be dangerous.
     */
    public void printEndangered(ArrayList<String> animalNames,
                                int dangerThreshold)
    {
        for(String animal : animalNames) {
            if(getCount(animal) <= dangerThreshold) {
                System.out.println(animal + " is endangered.");
            }
        }
    }

    public void printEndangeredStream(ArrayList<String> animalNames,
                                      int dangerThreshhold)
    {
        animalNames.stream()
        .filter ( animal -> getCount(animal) <= dangerThreshhold)
        .forEach ( animal -> System.out.println(animal + " is endangered."));
    }

    /**
     * Return a count of the number of sightings of the given animal.
     * @param animal The type of animal.
     * @return The count of sightings of the given animal.
     */
    public int getCount(String animal)
    {
        int total = 0;
        for(Sighting sighting : sightings) {
            if(animal.equals(sighting.getAnimal())) {
                total = total + sighting.getCount();
            }
        }
        return total;
    }

    /**
```

```
     * Remove from the sightings list all of those records with
     * a count of zero.
     */
    public void removeZeroCounts()
    {
        Iterator<Sighting> it = sightings.iterator();
        while(it.hasNext()) {
            Sighting record = it.next();
            if(record.getCount() == 0) {
                it.remove();
            }
        }
    }

    /**
     * Return a list of all sightings of the given type of animal
     * in a particular area.
     * @param animal The type of animal.
     * @param area The ID of the area.
     * @return A list of sightings.
     */
    public ArrayList<Sighting> getSightingsInArea(String animal, int area)
    {
        ArrayList<Sighting> records = new ArrayList<>();
        for(Sighting record : sightings) {
            if(animal.equals(record.getAnimal())) {
                if(record.getArea() == area) {
                    records.add(record);
                }
            }
        }
        return records;
    }

    /**
     * Return a list of all the sightings of the given animal.
     * @param animal The type of animal.
     * @return A list of all sightings of the given animal.
     */
    public ArrayList<Sighting> getSightingsOf(String animal)
    {
        ArrayList<Sighting> filtered = new ArrayList<>();
        for(Sighting record : sightings) {
            if(animal.equals(record.getAnimal())) {
                filtered.add(record);
            }
        }
        return filtered;
    }

}
```

3.2 Pre-Lab

Kubilay



Pre Lab - Lamda Functions - Kubilay Günther

(P1) ~~for(~~ If you have a collection called myList what Java code would you have to write to apply some code to each of the members in the List?

```
for (member : myList)
  { System.out.println (member); }
```

(P2) Given the code `public void printStudent (Student s) { System.out.println (s.getDetails()); }`, What is the equivalent Lamda in Java?

~~public void print Student~~ ⟸ Bt richtig

```
Student s ⟶ { System.out.println (s.getDetails()); }
```

~~⇒ [in Blue]~~

```
s ⟶ { System.out.println(s.getDetails()); }
```

(P3) Write pseudocode for determining how many elephants a particular spotter saw on ~~sighting~~ a particular day.

```
(sightings.filter (name == elephant).map (count).reduce (add);)
```

(in `map`, ~~number~~ is replaced by "count")

or.

✓

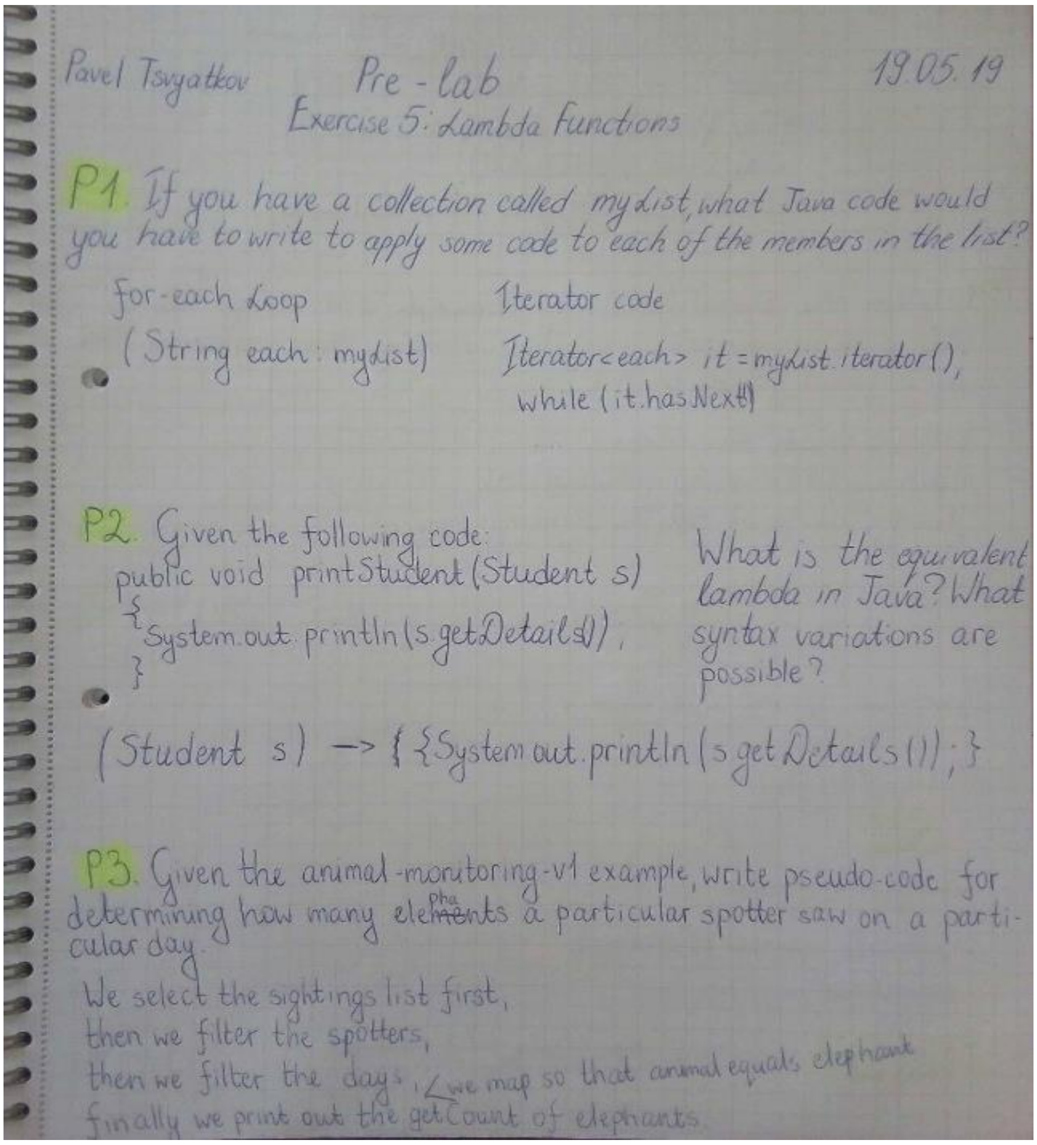
```
sightings.stream ()
      .filter (s → "Elephant".equals (s.getAnimals())
      .map    (s → "~~elephant" sighting~~ s.getAnimals())
      .forEach ( animals → System.out.println (details));
```

P4) Write pseudo-code to create a stream containing only those sightings that have a count greater than 0.

```
{    sighting.stream()
        .filter( s → animal
        .map (s → s.getCount())
        .reduce (0, (total, count) → total + count);
}
```

S. Map

Pavel


Pavel Tsvyatkov — Pre - lab — 19.05.19

Exercise 5: Lambda Functions

P1. If you have a collection called myList, what Java code would you have to write to apply some code to each of the members in the list?

for-each Loop
(String each : myList)

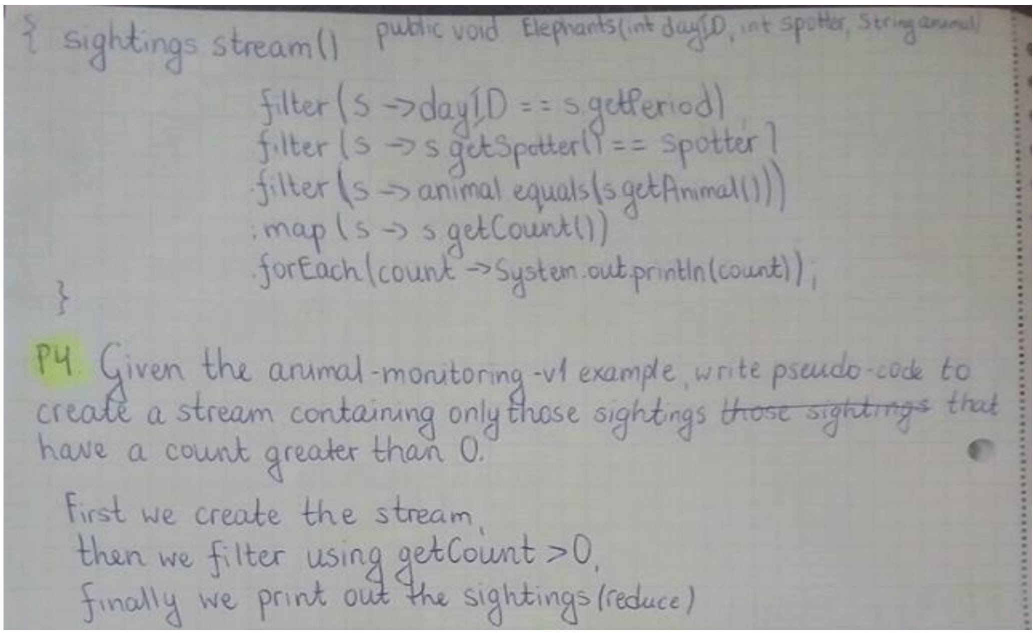
Iterator code
```
Iterator<each> it = myList.iterator();
while (it.hasNext())
```

P2. Given the following code:
```
public void printStudent (Student s)
{
  System.out.println(s.getDetails());
}
```
What is the equivalent lambda in Java? What syntax variations are possible?

```
(Student s) -> { {System.out.println(s.getDetails()); }
```

P3. Given the animal-monitoring-v1 example, write pseudo-code for determining how many ele~~ments~~pha a particular spotter saw on a particular day.

We select the sightings list first,
then we filter the spotters,
then we filter the days, ∠ we map so that animal equals elephant
finally we print out the getCount of elephants.


```
{ sightings.stream()   public void Elephants(int dayID, int spotter, String animal)
          filter(s -> dayID == s.getPeriod)
          filter(s -> s.getSpotter() == Spotter)
          .filter(s -> animal.equals(s.getAnimal()))
          .map(s -> s.getCount())
          .forEach(count -> System.out.println(count));
}
```

P4. Given the animal-monitoring-v1 example, write pseudo-code to create a stream containing only those sightings ~~those sightings~~ that have a count greater than 0.

First we create the stream,
then we filter using getCount > 0,
finally we print out the sightings (reduce)